Joc d'instruccions

Miquel Albert Orenga
Gerard Enrique Manonellas

PID_00218250

Universitat Oberta de Catalunya

Índex

Introducció..... 5

Objectius..... 6

1. Joc d'instruccions..... 7
1.1. Cicle d'execució 7
1.2. Arquitectura del joc d'instruccions 8
1.3. Representació del joc d'instruccions 9
1.4. Format de les instruccions 10
1.4.1. Elements que componen una instrucció 10
1.4.2. Mida de les instruccions 11
1.5. Operands 13
1.5.1. Nombre d'operands 13
1.5.2. Localització dels operands 14
1.5.3. Tipus i mida dels operands 15
1.6. Tipus d'instruccions 17
1.6.1. Bits de resultat 17
1.6.2. Instruccions de transferència de dades 18
1.6.3. Instruccions aritmètiques 19
1.6.4. Instruccions lògiques 23
1.6.5. Instruccions de ruptura de seqüència 26
1.6.6. Instruccions d'entrada/sortida 31
1.6.7. Altres tipus d'instruccions 32
1.7. Disseny del joc d'instruccions 32

2. Modes d'adreçament..... 34
2.1. Adreçament immediat 36
2.2. Adreçament directe 37
2.3. Adreçament indirecte 38
2.4. Adreçament relatiu 41
2.4.1. Adreçament relatiu a registre base 41
2.4.2. Adreçament relatiu a registre índex 42
2.4.3. Adreçament relatiu a PC 45
2.5. Adreçament implícit 46
2.5.1. Adreçament a pila 46

Resum..... 49

Introducció

El joc d'instruccions és el punt de trobada entre el dissenyador del computador i el programador. Des del punt de vista del dissenyador, és un pas més en l'explicitació del processador d'un computador i, des del punt de vista del programador en assemblador, l'eina bàsica per a accedir als recursos disponibles del computador.

Per a definir un joc d'instruccions cal saber com s'executen les instruccions, quins elements les formen i quines són les diferents maneres d'accedir a les dades i a les instruccions. Aquests són els conceptes que tractarem en aquest mòdul i ho farem de manera genèrica, per tal poder-los aplicar a tota mena de problemes reals i de processadors comercials.

Veurem com aquests conceptes afecten els diferents paràmetres de l'arquitectura del computador a l'hora de definir el joc d'instruccions, quines restriccions ens imposen i com podem arribar a un compromís entre facilitar la tasca al programador i donar una eficiència màxima al computador.

Objectius

Amb els materials didàctics d'aquest mòdul es pretén que els estudiants assoleixin els objectius següents:

1. Saber com es defineix el joc d'instruccions d'un processador.

2. Conèixer els tipus d'instruccions més habituals en el joc d'instruccions d'un computador de propòsit general.

3. Saber com funciona cadascuna de les instruccions del joc d'instruccions.

4. Aprendre les nocions bàsiques necessàries per a utilitzar un joc d'instruccions de baix nivell.

5. Familiaritzar-se amb les maneres de referenciar una dada en una instrucció.

1. Joc d'instruccions

La majoria de programes s'escriuen en llenguatges d'alt nivell, com ara C++, Java o Pascal. Per a poder executar un programa escrit en un llenguatge d'alt nivell en un processador, aquest programa s'ha de traduir primer a un llenguatge que pugui entendre el processador, diferent per a cada família de processadors. El conjunt d'instruccions que formen aquest llenguatge es denomina *joc d'instruccions* o *repertori d'instruccions*.

Per a poder definir un joc d'instruccions caldrà conèixer bé l'arquitectura del computador per a treure'n el màxim rendiment; en aquest mòdul veurem com queda definit un joc d'instruccions segons una arquitectura genèrica.

1.1. Cicle d'execució

Executar un programa consisteix a executar una seqüència d'instruccions, de manera que cada instrucció duu a terme un cicle d'execució. Aquesta seqüència d'instruccions no serà la seqüència escrita que fem d'un programa, sinó la seqüència temporal d'execució de les instruccions considerant les instruccions de salt.

El **cicle d'execució** és la seqüència d'operacions que es fan per a executar cada una de les instruccions i el dividirem en quatre fases principals:

1) Lectura de la instrucció
2) Lectura dels operands font
3) Execució de la instrucció i emmagatzematge de l'operand destinació
4) Comprovació d'interrupcions

Diferències en les fases del cicle d'execució

Les fases del cicle d'execuació són comunes a la majoria de computadors actuals i les diferències principals es troben en l'ordre en què es fan algunes de les operacions de cada fase o en què es fan algunes de les operacions en altres fases.

En la taula següent podeu veure de manera esquemàtica les operacions que es fan en cada fase:

Inici del cicle d'execució	
Fase 1 Lectura de la instrucció	Llegir la instrucció
	Descodificar la instrucció
	Actualitzar el PC

Inici del cicle d'execució	
Fase 2 Lectura dels operands font	Calcular l'adreça i llegir el primer operand font
	Calcular l'adreça i llegir el segon operand font
Fase 3 Execució de la instrucció i emmagatzematge de l'operand destinació	Executar la instrucció.
Fase 4 Comprovació d'interrupcions	Comprovar si algun dispositiu ha sol·licitat una interrupció.

1.2. Arquitectura del joc d'instruccions

Les instruccions d'una arquitectura són autocontingudes; és a dir, inclouen tota la informació necessària per a la seva execució.

Un dels elements necessaris en qualsevol instrucció és el **conjunt d'operands**. Els operands necessaris per a executar una instrucció poden trobar-se explícitament en la instrucció o poden ser implícits.

La localització dins del processador dels operands necessaris per a executar una instrucció i la manera d'explicitar-los donen lloc a arquitectures diferents del joc d'instruccions.

Segons els criteris de localització i l'explicitació dels operands, podem identificar les arquitectures del joc d'instruccions següents:

- **Arquitectures basades en pila**: els operands són implícits i es troben a la pila.
- **Arquitectures basades en acumulador**: un dels operands es troba de manera implícita en un registre anomenat *acumulador*.
- **Arquitectures basades en registres de propòsit general**: els operands es troben sempre de manera explícita ja sigui en registres de propòsit general o en la memòria.

Dins les arquitectures basades en registres de propòsit general podem distingir tres subtipus:

- **Registre-registre (o *load-store*)**. Només poden accedir a la memòria instruccions de càrrega (*load*) i emmagatzemament (*store*).
- **Registre-memòria**. Qualsevol instrucció pot accedir a la memòria amb un dels seus operands.

- **Memòria-memòria**. Qualsevol instrucció pot accedir a la memòria amb tots els seus operands.

Arquitectura memòria-memòria

L'arquitectura memòria-memòria és un tipus d'arquitectura que pràcticament no s'utilitza en l'actualitat.

Avantatges dels subtipus d'arquitectures

El subtipus registre-registre presenta els avantatges següents respecte dels registre-memòria i memòria-memòria: la codificació de les instruccions és molt simple i de longitud fixa, i utilitza un nombre semblant de cicles de rellotge del processador per a executar-se. Com a contrapartida, calen més instruccions per a fer la mateixa operació que en els altres dos subtipus.

Els subtipus registre-memòria i memòria-memòria presenten els avantatges següents respecte del registre-registre: no cal carregar les dades a registres per a operar-hi i faciliten la programació. Com a contrapartida, els accessos a memòria poden crear colls d'ampolla i el nombre de cicles de rellotge del processador necessaris per a executar una instrucció pot variar molt, cosa que dificulta el control del cicle d'execució i n'alenteix l'execució.

Podem parlar de cinc tipus d'arquitectura del joc d'instruccions:

1) Pila
2) Acumulador
3) Registre-registre
4) Registre-memòria
5) Memòria-memòria

Presentem gràficament les diferències entre aquests tipus d'arquitectura veient com resolen una mateixa operació: sumar dos valors (A i B) emmagatzemats en memòria i guardar el resultat en una tercera posició de memòria (C).

Pila	Acumulador	Registre-registre	Registre-memòria	Memòria-memòria
PUSH [A]	LOAD [A]	LOAD R1, [A]	MOV R1, [A]	MOV [C], [A]
PUSH [B]	ADD [B]	LOAD R2, [B]	ADD R1, [B]	ADD [C], [B]
ADD	STORE [C]	ADD R2, R1	MOV [C], R1	
POP [C]		STORE [C], R2		

Hem suposat que en les instruccions de dos operands, el primer operand actua com a operand font i operand destinació en el ADD i només com a operand destinació en la resta, el segon operand actua sempre com a operand font. A, B i C són les etiquetes que representen les adreces de memòria on hi ha emmagatzemats els valors que volem sumar i el seu resultat, respectivament. Els claudàtors indiquen que agafem el contingut d'aquestes adreces de memòria.

1.3. Representació del joc d'instruccions

El joc d'instruccions d'una arquitectura es representa des de dos punts de vista:

1) Des del punt de vista del computador, cada instrucció es representa com una seqüència de bits que es divideix en camps, en què cada camp correspon a un dels elements que formen la instrucció. Cada bit és la representació d'un senyal elèctric alt o baix. Aquesta manera de representar el joc d'instruccions s'acostuma a anomenar *codi de màquina* o *llenguatge de màquina*.

2) Des del punt de vista del programador, les instruccions es representen mitjançant símbols i s'utilitzen expressions mnemotècniques o abreviatures. Cal tenir present que és molt difícil treballar amb les representacions binàries pròpies del codi de màquina. Aquesta manera de representar el joc d'instruccions s'acostuma a anomenar *de màquina i d'assemblador* o *llenguatge d'assemblador*.

Arquitectura	Instrucció	Operació
CISCA	ADD R1, R2	Suma
Intel x86-64	MUL RAX	Multiplicació
MIPS	lw $S1, 100($S2)	Carrega en un registre el valor emmagatzemat en una posició de memòria

Exemple per a l'arquitectura CISCA

Instrucció per a sumar dos nombres que estan emmagatzemats en els registres R1 i R2. El resultat quedarà emmagatzemat a R1.

Des del punt de vista del programador: *codi d'assemblador*

Codi d'operació	Operand 1	Operand 2
ADD	R1	R2

Des del punt de vista del computador: *codi de màquina*

Codi d'operació	Operand 1	Operand 2
20h	11h	12h
0010 0000	0001 0001	0001 0010
8 bits	8 bits	8 bits
← 24 bits →		

Vegeu també

L'arquitectura CISCA és un model senzill de màquina definida en el mòdul 7 d'aquesta assignatura.

1.4. Format de les instruccions

1.4.1. Elements que componen una instrucció

Els elements que componen una instrucció, independentment del tipus d'arquitectura, són els següents:

- **Codi d'operació**: especifica l'operació que fa la instrucció.
- **Operand font**: per a fer l'operació poden ser necessaris un o més operands font; un o més d'aquests operands poden ser implícits.

- **Operand destinació**: emmagatzema el resultat de l'operació realitzada. Pot ser explícit o implícit. Un dels operands font es pot utilitzar també com a operand destinació.

- **Adreça de la instrucció següent**: especifica on és la instrucció següent que s'ha d'executar; acostuma a ser una informació implícita ja que el processador va a buscar automàticament la instrucció que es troba a continuació de la darrera instrucció executada. Només les instruccions de ruptura de seqüència especifiquen una adreça alternativa.

1.4.2. Mida de les instruccions

Un dels aspectes més importants a l'hora de dissenyar el format de les instruccions és determinar-ne la mida. En aquest sentit, trobem dues alternatives:

- **Instruccions de mida fixa**: totes les instruccions ocuparan el mateix nombre de bits. Aquesta alternativa simplifica el disseny del processador i l'execució de les instruccions pot ser més ràpida.

- **Instruccions de mida variable**: la mida de les instruccions dependrà del nombre de bits necessari per a cada una. Aquesta alternativa permet dissenyar un conjunt ampli de codis d'operació, l'adreçament pot ser més flexible i permet posar referències a registres i memòria. Com a contrapartida, augmenta la complexitat del processador.

> És desitjable que la mida de les instruccions sigui múltiple de la mida de la paraula de memòria.

Per a determinar la mida dels camps de les instruccions farem servir:

1) Codi d'operació. La tècnica més habitual és assignar un nombre fix de bits de la instrucció per al codi d'operació i reservar la resta de bits per a codificar els operands i els modes d'adreçament.

Exemple

Si es destinen N bits per al codi tindrem disponibles 2^N codis d'operació diferents; és a dir, 2^N operacions diferents.

En instruccions de mida fixa, com més bits es destinin al codi d'operació, menys en quedaran per als modes d'adreçament i la representació dels operands.

Es pot ampliar el nombre d'instruccions diferents en el joc d'instruccions sense afegir més bits al camp del codi d'operació. Només cal afegir-hi un camp nou, que anomenarem *expansió del codi d'operació*. Evidentment, aquest camp només s'afegeix a les instruccions en què es faci ampliació de codi.

Si dels 2^N codis de què disposem en reservem x per a fer l'expansió de codi i el camp d'expansió és de k bits, tindrem 2^k instruccions addicionals per cada codi reservat. D'aquesta manera, en lloc de tenir un total de 2^N instruccions, en tindrem $(2^n - x) + x \cdot 2^k$.

Exemple

Tenim $n = 3$, $x = 4$ i $k = 2$. En lloc de disposar de $2^3 = 8$ instruccions diferents, en tindrem $(2^3 - 4) + 4 \cdot 2^2 = 20$.

Per a fer l'expansió del codi d'operació hem de triar 4 codis. En aquest cas hem triat els codis (010, 011, 100, 101).

Codis d'operació de 3 bits		Codis d'operació amb expansió del codi	
		000	100 00
000		001	100 01
001		010 00	100 10
010		010 01	100 11
011		010 10	101 00
100		010 11	101 01
101		011 00	101 10
110		011 01	101 11
111		011 10	110
		011 11	111

Evidentment en les instruccions de mida fixa, les instruccions que utilitzin el camp d'expansió del codi d'operació disposaran de menys bits per a codificar la resta de camps de la instrucció.

2) Operands i modes d'adreçament. Un cop fixats els bits del codi d'operació podem assignar els bits corresponents als operands i als modes d'adreçament. Les dues maneres més habituals d'indicar quin mode d'adreçament utilitza cada un dels operands de la instrucció són les següents:

a) Amb el codi d'operació: aquesta tècnica s'utilitza habitualment quan no es poden utilitzar tots els modes d'adreçament en totes les instruccions. Aquesta tècnica és la utilitzada principalment en les arquitectures PowerPC, MIPS i SPARC.

b) En un camp independent: d'aquesta manera, per a cadascun dels operands afegim un camp per a indicar quin mode d'adreçament utilitza. La mida d'aquest camp dependrà del nombre de modes d'adreçament que tinguem i de si es poden utilitzar tots els modes en tots els operands. Aquesta tècnica és la utilitzada principalment en les arquitectures Intel x86-64 i VAX.

Per a codificar els operands segons els modes d'adreçament que utilitzen cal considerar una sèrie de factors:

a) Nombre d'operands de la instrucció: quants operands tenen les instruccions i si són sempre de la mateixa mida.

b) Ús de registres o de memòria com a operands: quants accessos a registres i a memòria hi pot haver en una instrucció.

c) Nombre de registres del processador.

d) Rang d'adreces del computador: quants bits es necessiten per a especificar una adreça que pot ser completa o parcial (com en el cas de memòria segmentada).

e) Nombre de bits per a codificar els camps de desplaçament o valors immediats.

Especificitats de l'arquitectura CISCA

Les següents són les especificitats de l'arquitectura CISCA:

- El nombre d'operands pot ser 0, 1 o 2 i de mida variable.
- No pot haver-hi dos operands que facin referència a memòria.
- Disposa d'un banc de 16 registres.
- Memòria de 2^{32} adreces de tipus byte (4 Gbytes). Necessitem 32 bits per a codificar les adreces.
- Els desplaçaments es codifiquen utilitzant 16 bits.
- Els valors immediats es codifiquen utilitzant 32 bits.

1.5. Operands

1.5.1. Nombre d'operands

Les instruccions poden utilitzar un nombre diferent d'operands segons el tipus d'instrucció de què es tracti.

Exemple

Cal tenir present que una mateixa instrucció en màquines diferents pot utilitzar un nombre diferent d'operands segons el tipus d'arquitectura del joc d'instruccions que utilitzi la màquina.

La instrucció aritmètica de suma (C = A + B) utilitza dos operands font (A i B) i produeix un resultat que s'emmagatzema en un operand destinació (C):

- En una arquitectura basada en pila, els dos operands font es trobaran al cim de la pila i el resultat s'emmagatzemarà també a la pila.
- En una arquitectura basada en acumulador, un dels operands font es trobarà al registre acumulador, l'altre serà explícit en la instrucció i el resultat s'emmagatzemarà a l'acumulador.
- En una arquitectura basada en registres de propòsit general els dos operands font seran explícits. L'operand destinació podrà ser un dels operands font (instruccions de dos operands) o un operand diferent (instruccions de tres operands).

Segons l'arquitectura i el nombre d'operands de la instrucció podem tenir diferents versions de la instrucció de suma, tal com es veu a la taula següent.

Pila	ADD	Suma els dos valors de dalt de la pila
Acumulador	ADD R1	Acumulador = Acumulador + R1
Registre-registre	ADD R1, R2	R1 = R1 + R2
	ADD R3, R1, R2	R3 = R1 + R2
Registre-memòria	ADD R1, [A01Bh]	R1 = R1+ M(A01Bh)
	ADD R2, R1, [A01Bh]	R2 = R1+ M(A01Bh)
Memòria-memòria	ADD [A01Dh], [A01Bh]	M(A01Dh) = M(A01Dh) + M(A01Bh)

Hem suposat que, en les instruccions de dos operands, el primer operand actua com a operand font i també com a operand destinació. Els valors entre claudàtors expressen una adreça de memòria.

1.5.2. Localització dels operands

Els operands representen les dades que hem d'utilitzar per a executar una instrucció. Aquestes dades es poden trobar en llocs diferents dins del computador. Segons la localització podem classificar els operands de la següent manera:

- **Immediat**. La dada està representada en la instrucció mateix. Podem considerar que es troba en un registre, el registre IR (*registre d'instrucció*), i està directament disponible per al processador.
- **Registre**. La dada estarà directament disponible en un registre dins del processador.
- **Memòria**. El processador haurà d'iniciar un cicle de lectura/escriptura a memòria.

En el cas d'operacions d'E/S, caldrà sol·licitar la dada al mòdul d'E/S adient i per a accedir als registres del mòdul d'E/S segons el mapa d'E/S que tinguem definit.

Vegeu també

En el mòdul "Sistemes d'entrada/sortida", veurem que als registres del mòdul d'E/S, anomenats *ports d'E/S*, hi podem accedir com si fossin posicions de memòria.

Segons el lloc on sigui la dada, el processador haurà de fer tasques diferents per tal d'obtenir-la: pot ser necessari fer càlculs i accessos a memòria indicats pel *mode d'adreçament* que utilitza cada operand.

Vegeu també

Els modes d'adreçament s'estudien a l'apartat 2 d'aquest mateix mòdul didàctic.

1.5.3. Tipus i mida dels operands

Els operands de les instruccions serveixen per a expressar el lloc on són les dades que hem d'utilitzar. Aquestes dades s'emmagatzemen com una seqüència de bits i, segons la interpretació dels valors emmagatzemats, podem tenir tipus de dades diferents que, generalment, també ens determinaran la mida dels operands. En molts jocs d'instruccions, el tipus de dada que utilitza una instrucció és determinat pel codi d'operació de la instrucció.

Hem de tenir present que una mateixa dada pot ser tractada com un valor lògic, com un valor numèric o com un caràcter, segons l'operació que s'hi faci, cosa que no passa amb els llenguatges d'alt nivell.

A continuació presentem els tipus generals de dades més habituals:

1) Adreça. Tipus de dada que expressa una adreça de memòria. Per a operar amb aquest tipus de dada, pot ser necessari efectuar càlculs i accessos a memòria per a obtenir l'adreça efectiva.

Vegeu també

Tractarem la manera d'operar amb les adreces en l'apartat 2 d'aquest mateix mòdul didàctic.

La manera d'expressar i codificar les adreces dependrà de la manera d'accedir a la memòria del computador i dels modes d'adreçament que suporti el joc d'instruccions.

2) Nombre. Tipus de dada que expressa un valor numèric. Habitualment distingim tres tipus de dades numèriques (i cada un d'aquests tipus es pot considerar amb signe o sense signe):

a) Nombres enters
b) Nombres en punt fix
c) Nombres en punt flotant

Com que disposem d'un espai limitat per a expressar valors numèrics, tant la magnitud dels nombres com la seva precisió també ho seran, cosa que limitarà el rang de valors que podem representar.

Segons el joc d'instruccions amb què es treballa, en el codi d'assemblador, es poden expressar els valors numèrics de maneres diferents: en decimal, hexadecimal, binari, utilitzant punts o comes (si és en punt fix) i amb altres sintaxis per a nombres en punt flotant; però en codi de màquina sempre els codifica-

rem en binari utilitzant un tipus de representació diferent per a cada tipus de valor. Cal dir, però, que un dels més utilitzats és el conegut com a *complement a 2*, que és una representació en punt fix de valors amb signe.

Exemple

Si tenim 8 bits i la dada és un enter sense signe, el rang de representació serà [0,255], i si la dada és un enter amb signe, utilitzant complement a 2 el rang de representació serà [–128,127].

Forma d'expressar-ho		Forma de codificar-ho amb 8 bits	
En decimal	En hexadecimal	Enter sense signe	Enter amb signe a Ca2
12	0Ch	0000 1100	0000 1100
–33	DFh	No es pot (té signe)	1101 1111
150	96h	1001 0110	No es pot (fora de rang)

3) Caràcter. Tipus de dada que expressa un caràcter. Habitualment s'utilitza per a formar cadenes de caràcters que representaran un text.

Tot i que el programador representa els caràcters mitjançant les lletres de l'alfabet, per a poder representar-los en un computador que utilitza dades binàries caldrà una codificació. La codificació més habitual és l'ASCII, que representa cada caràcter amb un codi de 7 bits, cosa que permet obtenir 128 caràcters diferents. Els codis ASCII s'emmagatzemen i es transmeten utilitzant 8 bits per caràcter (1 byte), el vuitè bit dels quals té la funció de paritat o control d'errors.

El vuitè bit

En els sistemes més actuals s'utilitza el vuitè bit per a ampliar el joc de caràcters i disposar de símbols addicionals, com ara lletres que no existeixen en l'alfabet anglès (*ç*, *ñ*, etc.), lletres amb accent, dièresi, etc.

Hi ha altres codificacions que es poden utilitzar sobretot en llenguatges d'alt nivell, com per exemple Unicode, que fa servir 16 bits i és molt interessant per a donar suport multilingüe.

4) Dada lògica. Tipus de dada que expressa un conjunt de valors binaris o booleans; generalment cada valor s'utilitza com una unitat, però pot ser interessant tractar-los com a cadenes de *N* bits en què cada element de la cadena és un valor booleà, un bit que pot valer 0 o 1. Aquest tractament és útil quan s'utilitzen instruccions lògiques com AND, OR o XOR.

Això també permet emmagatzemar en un sol byte fins a 8 valors booleans que poden representar diferents dades, en lloc d'utilitzar més espai per a representar aquests mateixos valors.

1.6. Tipus d'instruccions

En general, els codis d'operació d'un joc d'instruccions varien d'una màquina a una altra, però podem trobar els mateixos tipus d'instruccions en gairebé totes les arquitectures i els podem classificar de la manera següent:

- Transferència de dades
- Aritmètiques
- Lògiques
- Transferència del control
- Entrada/sortida
- Altres tipus

1.6.1. Bits de resultat

Els bits de resultat ens donen informació de com és el resultat obtingut en operació realitzada a la unitat aritmeticològica (ALU) del processador. En executar una instrucció aritmètica o lògica, la unitat aritmeticològica fa l'operació i obté un resultat; segons el resultat activa els bits de resultat que correspongui i s'emmagatzemen en el registre d'estat per a poder ser utilitzats per altres instruccions. Els bits de resultat més habituals són els següents:

Nota

Considerem que els bits de resultat són actius quan valen 1, i inactius quan valen 0.

- **Bit de zero (Z)**: s'activa si el resultat obtingut és 0.

- **Bit de transport (C)**: també anomenat *carry* a la suma i *borrow* a la resta. S'activa si en el darrer bit que operem en una operació aritmètica es produeix transport, també pot ser degut a una operació de desplaçament. S'activa si al final de l'operació en portem una segons l'algorisme de suma i resta tradicional operand en binari o si el darrer bit que desplacem es copia sobre el bit de transport i aquest és 1.
  Quan operem amb nombres enters sense signe, el bit de transport és equivalent al bit de sobreeiximent; però amb nombre amb signe (com és el cas del Ca2), el bit de transport i el bit de sobreeiximent no són equivalents i la informació que aporta el bit de transport sobre el resultat no és rellevant.

  Bit de transport a la resta

  El bit de transport, també anomenat *borrow*, es genera segons l'algorisme convencional de la resta. Però si per a fer la resta A – B, ho fem sumant el complementari del subtrahend, A + (–B), el bit de transport de la resta (*borrow*) serà el bit de transport (*carry*) negat obtingut de fer la suma amb el complementari, (*borrow* = *carry negat*), llevat dels casos en què B = 0, en què aleshores tant el *carry* com el *borrow* són iguals i valen 0.

- **Bit de sobreeiximent (V)**: també anomenat *overflow*. S'activa si l'última operació ha produït sobreeiximent segons el rang de representació utilitzat. Per a representar el resultat obtingut, en el format que estem utilitzant, necessitaríem més bits dels disponibles.

- **Bit de signe (S)**: actiu si el resultat obtingut és negatiu, el bit més significatiu del resultat és 1.

1.6.2. Instruccions de transferència de dades

Les instruccions de transferència de dades transfereixen una dada d'una localització a una altra dins el computador.

El tipus i la mida de les dades poden estar especificats de manera implícita en el codi d'operació. Això farà que tinguem codis d'operació diferents segons el tipus i la mida de les dades que s'han de transferir i per a indicar implícitament la localització de l'operand font o destinació.

Mida dels operands

Normalment, cada adreça de memòria es correspon amb una posició de memòria de mida 1 byte. Si fem STORW [1000], R1, i R1 és un registre de 16 bits, en realitat estem utilitzant dues adreces de memòria, la 1000 i la 1001, ja que la dada ocupa 2 bytes a R1.

STORx: Indica que transferim una dada cap a la memòria.

STORB [1000], 020h: Indica que transferim 1 byte a la posició de memòria 1000.

STORW [1000], R1: Indica que transferim 2 bytes (1 word), el contingut del registre R1, a les posicions de memòria 1000 i 1001.

En les arquitectures de tipus registre-registre s'utilitzen les instruccions LOAD i STORE quan cal fer transferències de dades amb la memòria.

Exemple d'instruccions de transferència de dades en l'arquitectura MIPS

Instrucció	Exemple	Operació
LOAD destinació, font	LW $s1,100($s2)	Mou el contingut de la posició de memòria M($s2+100) al registre $s1
STORE font, destinació	SW $s1,100($s2)	Mou el contingut del registre $s1 a la posició de memòria M($s2+100)
MOVE destinació, font	MOVE $s1,$s2	Mou el contingut del registre $s2 al registre $s1

En les arquitectures de tipus registre-memòria o memòria-memòria s'utilitzen instruccions MOV quan cal fer transferències de dades amb la memòria. És el cas de les arquitectures Intel x86-64 i CISCA.

Exemple d'instruccions de transferència de dades en l'arquitectura Intel x86-64

Instrucció	Operació	Exemple
MOV destinació, font	destinació ← font	MOV RAX, [nom_var]
XCHG destinació, font	destinació ←→ font Intercanvia el contingut dels dos operands.	XCHG [nom_var], RBX
POP destinació	destinació ← M(SP), actualitza SP. Treu el valor que hi ha al cim de la pila.	POP RAX

Exemple d'instruccions de transferència de dades en l'arquitectura CISCA

Instrucció	Operació	Exemple
MOV destinació, font	destinació ← font	MOV R1, [nom_var]
PUSH font	Actualitza SP, M(SP) ← font. Posa el valor al cim de la pila.	PUSH R5

1.6.3. Instruccions aritmètiques

Totes les màquines inclouen instruccions aritmètiques bàsiques (suma, resta, multiplicació i divisió) i d'altres com negar, incrementar, decrementar o comparar.

Quan fem operacions aritmètiques hem de tenir presents els tipus d'operands i si els hem de considerar nombres amb signe o sense signe, ja que en algunes operacions, com la multiplicació o la divisió, això pot donar lloc a instruccions diferents.

En aquest subapartat ens centrarem només en les operacions aritmètiques de nombres enters amb signe i sense signe. El format habitual per a representar nombres enters amb signe és el complement a 2.

Per a la representació de nombres decimals en punt fix es pot utilitzar la mateixa representació que per als enters fent que la coma binària estigui implícita en alguna posició del nombre. Per a la representació en punt flotant caldrà especificar un format diferent per a representar els nombres i instruccions específiques per a operar-hi.

Suma i resta

Aquestes operacions fan la suma i la resta de dos nombres; en alguns processadors es poden fer tenint en compte el valor del bit de transport com a bit de transport inicial.

Exemple de suma i resta en l'arquitectura CISCA

Instrucció	Operació	Exemple
ADD destinació, font	destinació = destinació + font	ADD R1, R2
SUB destinació, font	destinació = destinació – font	SUB R1, R2

Exemple de suma i resta en l'arquitectura Intel x86-64 considerant el bit de transport inicial

Instrucció	Operació	Exemple
ADC destinació, font	destinació = destinació + font + bit de transport	ADC RAX, RBX
SBB destinació, font	destinació = destinació – font – bit de transport	SBB RAX, RBX

Multiplicació

Multiplicació i operands

En la majoria d'arquitectures, l'operació multiplicació utilitza operands destinació implícits, d'aquesta manera és més fàcil representar un resultat de mida superior a la dels operands font.

Aquesta operació fa la multiplicació entera de dos nombres i cal tenir present que el resultat que es genera té una mida superior a la dels operands font.

Caldrà disposar de dues operacions diferents si es vol tractar operands amb signe o sense signe.

Exemple d'una instrucció de multiplicació en l'arquitectura Intel x86-64

Un operand font pot ser implícit i correspondre al registre AL (8 bits), al registre AX (16 bits), al registre EAX (32 bits) o al registre RAX (64 bits), l'altre operand font és explícit, i pot ser un operand de 8, 16, 32 o 64 bits.

- Si l'operand font és de 8 bits, l'operació que es fa és AX = AL * font, que amplia el resultat a un valor de 16 bits.
- Si l'operand font és de 16 bits, l'operació que es fa és DX,AX = AX * font, que amplia el resultat a un valor de 32 bits.
- Si l'operand font és de 32 bits, l'operació que es fa és EDX,EAX = EAX * font, que amplia el resultat a un valor de 64 bits.
- Si l'operand font és de 64 bits, l'operació que es fa és RDX,RAX = RAX * font, que amplia el resultat a un valor de 128 bits.

Instrucció	Exemple	Operació
MUL font (operació sense considerar el signe, un operand implícit)	MUL RBX	RDX,RAX = RAX * RBX
IMUL font (operació considerant el signe, un operant implícit)	IMUL BL	AX = AL * BL
IMUL destinació font (operació considerant el signe, dos operands explícits)	IMUL EAX, EBX	EAX = EAX * EBX

Exemple d'una instrucció de multiplicació en l'arquitectura CISCA

Els dos operands font i destinació són explícits, i el resultat s'emmagatzema en el primer operand que fa també d'operand destinació; tots els operands són nombres en complement a 2 de 32 bits; no s'utilitza cap operand implícit perquè no s'amplia la mida del resultat.

Instrucció	Operació	Exemple
MUL destinació, font	destinació = destinació * font	MUL R1, R2

Divisió

Divisió i operands

En la majoria d'arquitectures, l'operació divisió utilitza operands destinació implícits per a representar els dos resultats que s'obtenen i no perdre els valors dels operands font.

Aquesta operació fa la divisió entera de dos nombres i se n'obtenen dos resultats: el quocient i el residu de la divisió.

Caldrà disposar de dues operacions diferents si es vol tractar operands amb signe o sense signe.

Exemple d'una instrucció de divisió en l'arquitectura Intel x86-64

Divideix el dividend implícit entre el divisor explícit.

- Operand font de 8 bits: (quocient) AL = AX / font, (residu) AH = AX mod font
- Operand font de 16 bits: (quocient) AX = DX:AX / font, (residu) DX = DX:AX mod font
- Operand font de 32 bits: (quocient) EAX = EDX:EAX / font, (residu) EDX = EDX:EAX mod font
- Operand font de 64 bits: (quocient) RAX = RDX:RAX / font, (residu) RDX = RDX:RAX mod font

Instrucció	Exemple	Operació
DIV font (operació sense considerar el signe)	DIV RBX	RAX = RDX:RAX / RBX RDX = RDX:RAX mod RBX
IDIV font (operació considerant el signe)	IDIV EBX	EAX = EDX:EAX / EBX EDX = EDX:EAX mod EBX

Exemple d'una instrucció de divisió en l'arquitectura CISCA

Els dos operands font i destinació són explícits i són nombres amb signe. El quocient s'emmagatzema en el primer operand, i el residu, en el segon. Els dos operands fan d'operand font i destinació.

Instrucció	Exemple	Operació
DIV destinació, font DIV (dividend/quocient), (divisor/residu)	DIV R1, R2	R1 = R1 / R2 (quocient) R2 = R1 mod R2 (residu)

Increment i decrement

Aquestes operacions són un cas especial de la suma i la resta. S'acostumen a incloure en la majoria de jocs d'instruccions ja que són operacions molt freqüents: saber el valor (generalment, 1) que s'ha d'incrementar o decrementar facilita molt la seva implementació en el processador.

Exemple d'instruccions d'increment i decrement en l'arquitectura Intel x86-64

Instrucció	Operació	Exemple
INC destinació	destinació = destinació + 1	INC RAX
DEC destinació	destinació = destinació – 1	DEC BL

Exemple d'instruccions d'increment i decrement en l'arquitectura CISCA

Instrucció	Operació	Exemple
INC destinació	destinació = destinació + 1	INC R3
DEC destinació	destinació = destinació – 1	DEC R5

Comparació

Aquesta operació fa la comparació entre dos operands, restant el segon del primer i actualitzant els bits de resultat. És un cas especial de la resta en què el valor dels operands no es modifica perquè no se'n guarda el resultat.

Exemple d'instruccions de comparació en l'arquitectura Intel x86-64

Instrucció	Operació	Exemple
CMP destinació, font	destinació – font	CMP RAX, RBX

Exemple de comparació en l'arquitectura CISCA

Instrucció	Operació	Exemple
CMP destinació, font	destinació – font	CMP R1,R2

Negació

Aquesta operació canvia el signe de l'operand. Si les dades estan en complement a 2 és equivalent a fer una d'aquestes operacions: 0 – operand, NOT(operand) + 1, operand * (–1).

Exemple d'una instrucció de negació en l'arquitectura Intel x86-64

Instrucció	Operació	Exemple
NEG destinació	destinació = NOT(destinació) + 1	NEG EAX

Exemple d'una instrucció de negació en l'arquitectura CISCA

Instrucció	Operació	Exemple
NEG destinació	destinació = NOT(destinació) + 1	NEG R7

Totes les instruccions aritmètiques i de comparació poden modificar els bits de resultat.

Instrucció	Z	S	C	V
Suma	x	x	x	x
Resta	x	x	x	x
Multiplicació	x	x	–	x
Divisió	x	x	–	x
Increment	x	x	x	x
Decrement	x	x	x	x
Comparació	x	x	x	x

Aquests són els bits de resultat que es modifiquen habitualment, però això pot variar lleugerament en alguns processadors.
Notació: x indica que la instrucció *pot modificar aquest bit*. "–" indica que la instrucció *no modifica aquest bit*.

Instrucció	Z	S	C	V
Negació	x	x	x	x

Aquests són els bits de resultat que es modifiquen habitualment, però això pot variar lleugerament en alguns processadors.
Notació: x indica que la instrucció *pot modificar aquest bit*. "–" indica que la instrucció *no modifica aquest bit*.

1.6.4. Instruccions lògiques

Hi ha diverses instruccions lògiques:

1) Operacions lògiques. Les instruccions que fan operacions lògiques permeten manipular de manera individual els bits d'un operand. Les operacions lògiques més habituals són AND, OR, XOR, NOT.

Les instruccions lògiques fan l'operació indicada bit a bit; és a dir, el resultat que es produeix en un bit no afecta la resta.

x	y	x AND y	x OR y	x XOR y	NOT x
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

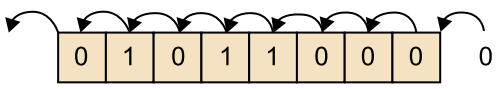
2) Operacions de desplaçament i rotació. Dins d'aquest grup d'instruccions s'inclouen instruccions de desplaçament lògic (SHL, SHR), desplaçament aritmètic (SAL, SAR), rotació (ROL, ROR).

Aquest grup d'instruccions inclou un operand amb el qual es fa l'operació i, opcionalment, un segon operand que indica quants bits s'han de desplaçar/rotar.

Ara veurem el funcionament de les instruccions lògiques de desplaçament i rotació:

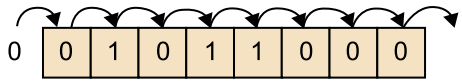
a) Desplaçament lògic a l'esquerra (SHL). Es considera el primer operand com un valor sense signe. Es desplacen els bits a l'esquerra tantes posicions com indica el segon operand; el bit de més a l'esquerra en determinades arquitectures es perd i en d'altres es copia sobre el bit de transport i s'hi afegeixen zeros per la dreta.

Exemple de desplaçament d'un bit a l'esquerra



b) Desplaçament lògic a la dreta (SHR). Es considera el primer operand com un valor sense signe; es desplacen els bits a la dreta tantes posicions com indica el segon operand, el bit de la dreta en determinades arquitectures es perd i en d'altres es copia sobre el bit de transport i s'hi afegeixen zeros per l'esquerra.

Exemple de desplaçament d'un bit a la dreta



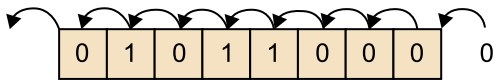
c) Desplaçament aritmètic a l'esquerra (SAL). Es considera el primer operand com un valor amb signe expressat en complement a 2; es desplacen els bits a l'esquerra tantes posicions com indica el segon operand, el bit de més a l'esquerra en determinades arquitectures es perd i en d'altres es copia sobre el bit de transport i s'hi afegeixen zeros per la dreta. Considerarem que hi ha sobreeiximent si el signe del resultat és diferent del signe del valor que desplacem.

Físicament, és la mateixa instrucció que la instrucció de desplaçament lògic a l'esquerra.

Nota

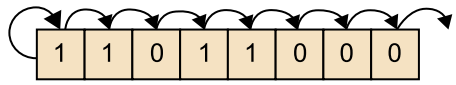
L'operació de desplaçament aritmètic a l'esquerra n posicions és equivalent a multiplicar el valor del primer operand per 2^n.

Exemple de desplaçament d'un bit a l'esquerra



d) Desplaçament aritmètic a la dreta (SAR). Es considera el primer operand com un valor amb signe expressat en complement a 2. El bit de més a l'esquerra, bit de signe, es conserva i es va copiant sobre els bits que es van desplaçant a la dreta. Es desplaça a la dreta tantes posicions com indica el segon operand i els bits de la dreta en determinades arquitectures es perden i en d'altres es copien sobre el bit de transport.

Exemple de desplaçament d'un bit a la dreta

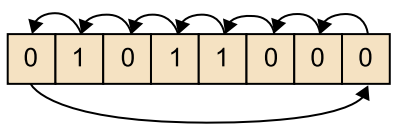


Nota

L'operació de desplaçament aritmètic a la dreta n posicions és equivalent a dividir el valor del primer operand entre 2^n.

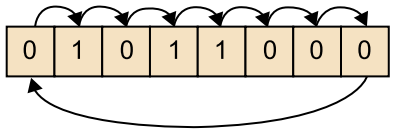
e) Rotació a l'esquerra (ROL). Es considera el primer operand com un valor sense signe; es desplacen els bits a l'esquerra tantes posicions com indica el segon operand, el bit de més a l'esquerra es passa a la posició menys significativa de l'operand.

Exemple de rotació d'un bit a l'esquerra



f) Rotació a la dreta (ROR). Es considera el primer operand com un valor sense signe; es desplacen els bits a la dreta tantes posicions com indica el segon operand, el bit de més a la dreta es passa a la posició més significativa de l'operand.

Exemple de rotació d'un bit a la dreta



Hi ha instruccions de rotació que utilitzen el bit de transport:

- Quan es rota a la dreta, el bit menys significatiu de l'operand es copia sobre el bit de transport i el bit de transport original es copia sobre el bit més significatiu de l'operand.
- Quan es rota a l'esquerra, el bit més significatiu de l'operand es copia sobre el bit de transport i el bit de transport es copia sobre el bit menys significatiu.

Les instruccions lògiques també poden modificar els bits de resultat del processador.

Instrucció	Z	S	C	V
AND	x	x	0	0
OR	x	x	0	0
XOR	x	x	0	0
NOT	–	–	–	–
Desplaçaments lògics	x	x	x	0
Desplaçaments aritmètics	x	x	x	x
Rotacions	–	–	x	x

Notació: x indica que la instrucció *pot modificar el bit de resultat*. – indica que la instrucció *no modifica el bit de resultat*. 0 indica que la instrucció *posa el bit de resultat a 0*.

Exemple d'instruccions lògiques de desplaçament i rotació en l'arquitectura Intel x86-64

Instrucció	Operació	Exemple
AND destinació, font	destinació = font AND destinació	AND AL,BL
OR destinació, font	destinació = font OR destinació	OR RAX,RBX
SHL destinació, font	destinació = destinació * 2^{font} (considerant destinació com un valor sense signe)	SHL AL, 4
SHR destinació, font	destinació = destinació / 2^{font} (considerant destinació com un valor sense signe)	SHR RAX, CL
RCL destinació, font	(CF = bit més significatiu de destinació, destinació = destinació * 2 + $CF_{original}$) farem això tants cops com indiqui l'operand font.	RCL EAX, CL
RCR destinació, font	(CF = bit menys significatiu de destinació, destinació = destinació / 2 + $CF_{original}$) farem això tants cops com indiqui l'operand font.	RCR RBX, 1

Exemple d'instruccions lògiques en l'arquitectura CISCA

Instrucció	Operació	Exemple
XOR destinació, font	destinació = font XOR destinació	XOR R2,R7
NOT destinació	destinació negat bit a bit	NOT R3
SAL destinació, font	destinació = destinació * 2^{font}	SAL R9, 3
SAR destinació, font	destinació = destinació / 2^{font}	SAR R9, 2

1.6.5. Instruccions de ruptura de seqüència

Les instruccions de ruptura de seqüència permeten canviar la seqüència d'execució d'un programa. En el cicle d'execució de les instruccions, el PC s'actualitza de manera automàtica apuntant a la instrucció emmagatzemada a continuació de la que s'està executant; per a poder-la canviar, cal saber quina és la següent instrucció que s'ha d'executar per a carregar-ne l'adreça al PC.

Entre les instruccions de ruptura de seqüència trobem les següents:

- Instruccions de salt o bifurcació
  - Instruccions de salt incondicional
  - Instruccions de salt condicional

- Instruccions de crida i retorn de subrutina

- Instruccions d'interrupció de programari i retorn d'una rutina de servei d'interrupció.

Instruccions de salt incondicional

Les instruccions de salt incondicional carreguem en el registre del PC l'adreça especificada per l'operand, que s'expressa com una etiqueta que representa l'adreça de la instrucció que s'ha d'executar.

Si es codifica l'operand com una adreça, actualitzarem el registre PC amb aquesta adreça. Si es codifica l'operand com un desplaçament haurem de sumar al PC aquest desplaçament i, com veurem més endavant, aquesta manera d'obtenir l'adreça de salt s'anomena *adreçament relatiu a PC*.

Exemple d'instruccions de ruptura de seqüència en l'arquitectura CISCA

Instrucció	Operació	Exemple
JMP etiqueta	[PC] ← etiqueta Salta a la instrucció indicada per l'etiqueta	JMP bucle

```
bucle: MOV R0, R1 ; Això és un bucle infinit
       JMP bucle
       MOV R2, 0 ; No s'executa mai
```

bucle és el nom de l'etiqueta que fa referència a l'adreça de memòria on s'emmagatzema la instrucció MOV R0, R1.

Instruccions de salt condicional

Les instruccions de salt condicional carreguen al registre PC l'adreça especificada per l'operand si es compleix una condició determinada (la condició és certa); en cas contrari, (la condició és falsa) el procés continua amb la instrucció següent de la seqüència. Aquest operand s'expressa com una etiqueta que representa l'adreça de la instrucció que s'ha d'executar, però habitualment es codifica com un desplaçament respecte del registre PC. Aquesta manera d'expressar una adreça de memòria, com veurem més endavant, s'anomena *adreçament relatiu a PC*.

Les condicions s'avaluen comprovant el valor actual dels bits de resultat, els més habituals són zero, signe, transport i sobreeiximent. Una condició pot avaluar un bit de resultat o més.

Les instruccions de salt condicional s'han d'utilitzar immediatament després d'una instrucció que modifiqui els bits de resultat, com ara les aritmètiques o lògiques, en altre cas, s'avaluaran els bits de resultat de l'última instrucció que els hagi modificat, situació gens recomanable.

La taula següent mostra els bits de resultat que cal comprovar per avaluar una condició determinada després d'una instrucció de comparació (CMP). Aquests bits de resultat s'activen segons els valors dels operands comparats.

Condició	Bits que comprova amb operands sense signe	Bits que comprova amb operands amb signe
A = B	Z = 1	Z = 1
A ≠ B	Z = 0	Z = 0
A > B	C = 0 i Z = 0	Z = 0 i S = V
A ≥ B	C = 0	S = V
A < B	C = 1	S ≠ V
A ≤ B	C = 1 o Z = 1	Z = 1 o S ≠ V

Nota

Cal tenir present que s'activarà un conjunt de bits diferent segons si els operands comparats es consideren amb signe o sense signe.

Exemple d'instruccions de salt condicional comunes a l'arquitectura Intel x86-64 i CISCA

Instrucció	Operació	Exemple
JE etiqueta (Jump Equal – Salta si igual)	Salta si Z = 1	JE eti3
JNE etiqueta (Jump Not Equal – Salta si diferent)	Salta si Z = 0	JNE eti3
JG etiqueta (Jump Greater – Salta si més gran)	Salta si Z = 0 i S = V (operands amb signe)	JG eti3
JGE etiqueta (Jump Greater or Equal – Salta si més gran o igual)	Salta si S = V (operands amb signe)	JGE eti3
JL etiqueta (Jump Less – Salta si més petit)	Salta si S ≠ V (operands amb signe)	JL eti3
JLE etiqueta (Jump Less or Equal – Salta si més petit o igual)	Salta si Z = 1 o S ≠ V (operands amb signe)	JLE eti3

```
      MOV R2, 0
eti3: INC R2 ; Això és un bucle de 3 iteracions
      CMP R2, 3
      JL eti3
      MOV R2, 7
```

Si considerem que la instrucció INC R2 és a la posició de memòria 0000 1000h, l'etiqueta eti3 farà referència a aquesta adreça de memòria, però quan codifiquem la instrucció JL eti3, no codificarem aquesta adreça sinó que codificarem el desplaçament respecte del PC. El desplaçament que hem de codificar és eti3 – PC.

Exemple d'instruccions de salt condicional específiques de l'arquitectura Intel x86-64

Instrucció	Operació	Exemple
JA etiqueta (Jump Above – Salta si superior)	Salta si C = 0 i Z = 0 (operands sense signe)	JA bucle
JAE etiqueta (Jump Above or Equal – Salta si superior o igual)	Salta si C = 0 (operands sense signe)	JAE bucle
JB etiqueta (Jump Below – Salta si inferior)	Salta si C = 1 (operands sense signe)	JB bucle
JBE etiqueta (Jump Below or Equal – Salta si inferior o igual)	Salta si C = 1 o Z = 1 (operands sense signe)	JBE bucle
JV etiqueta	Salta si V = 1	JV bucle

Instruccions de salt implícit

Hi ha un tipus d'instruccions de salt anomenades *skip* o *de salt implícit* que segons una condició salten una instrucció. Si no es compleix la condició executen la instrucció següent i si es compleix la condició no executen la instrucció següent, se la salten, i executen la instrucció que hi ha a continuació. En alguns casos també tenen una operació associada (com incrementar o decrementar un registre). Aquestes instruccions les solen tenir computadors molt simples o microcontroladors amb un joc d'instruccions reduït i amb instruccions de mida fixa.

Per exemple, tenim la instrucció *DSZ Registre* que decrementa *Registre* i salta si és zero.

```
iteracio:
        ...
        DSZ R1
```

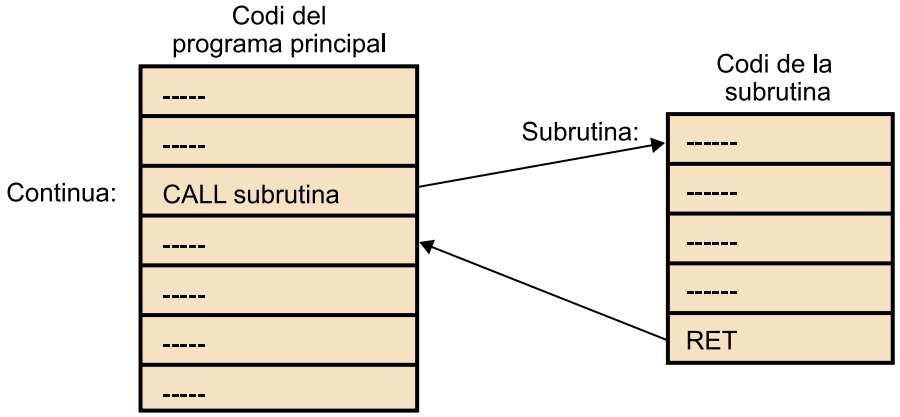
```
        JMP iteració
continuar:
        ...
```

Si R1 = 1 (en decrementar valdrà 0) anirà a l'etiqueta *continuar:*, sinó, executa el JMP (salt incondicional) i salta a l'etiqueta *iteracio:*.

Instruccions de crida i retorn de subrutina

Una subrutina és un conjunt d'instruccions que fa una funció concreta i que normalment es crida diverses vegades dins del programa principal.

Tant les instruccions de salt com les de crida i retorn a subrutina permeten trencar la seqüència d'execució d'un programa, però les darreres garanteixen la tornada al punt on s'ha trencat la seqüència un cop ha finalitzat l'execució.



La **crida a subrutina** és una instrucció que transfereix el control a la subrutina de manera que es pugui garantir el retorn al punt on es trobava la seqüència d'execució en cridar-la.

La instrucció té un sol operand que especifica l'adreça d'inici de la subrutina. Aquest operand s'expressa com una etiqueta que representa l'adreça de la primera instrucció de la subrutina que s'ha d'executar.

Les crides a subrutines duen a terme dues operacions:

1) Guardar el valor del PC en una localització coneguda perquè quan finalitzi l'execució de la subrutina pugui retornar al punt d'execució on es trobava i continuar la seqüència d'execució. Les localitzacions on es pot emmagatzemar el PC (adreça de retorn de la subrutina) són un registre, al principi de la subrutina o a la pila. En la majoria de computadors s'utilitza la pila per a emmagatzemar l'adreça de retorn.

2) Carregar en el PC l'adreça expressada per l'operand de la instrucció per a transferir el control a la subrutina.

El **retorn de subrutina** és una instrucció que s'utilitza per a retornar el control al punt d'execució des d'on s'ha fet la crida. Per a fer-ho, recupera el valor del PC del lloc on l'hagi emmagatzemat la instrucció de crida a subrutina. Ha de ser l'última instrucció d'una subrutina i no té cap operand explícit.

Cal que el programador s'asseguri que, abans d'executar el retorn de subrutina, en el cim de la pila hi hagi l'adreça de retorn. Per això és molt important fer una bona gestió de la pila dins les subrutines.

Exemple d'instruccions de crida i retorn de subrutina comunes a l'arquitectura Intel x86-64 i CISCA

Instrucció	Operació	Exemple
CALL etiqueta	Transfereix el control a la subrutina.	CALL subrutina
RET	Retorna de la subrutina	RET

Exemple de crida i un retorn de subrutina a l'arquitectura CISCA

En aquesta arquitectura, la pila s'implementa a memòria i creix cap a adreces baixes, per tant, en la instrucció CALL es decrementa SP (R15 en aquesta arquitectura) i en la instrucció RET s'incrementa. L'increment i el decrement que caldrà fer del registre SP és de 4 unitats, ja que cada posició de la pila emmagatzema 1 byte i l'adreça n'ocupa 4.

CALL 'nom_subrutina'

SP = SP – 4

M(SP) = PC

PC = Adreça inici 'nom-subrutina'

RET

PC = M(SP)

SP = SP + 4

Instruccions d'interrupció de programari i retorn d'una rutina de servei d'interrupció

La **interrupció de programari** és un tipus d'instrucció que implementa una interrupció cridant una rutina de servei d'interrupció (RSI). Aquests serveis generalment són rutines del sistema operatiu per a facilitar al programador les operacions d'E/S i d'accés a diferents elements del maquinari.

Aquest tipus d'instrucció té un sol operand que especifica un valor immediat que identifica el servei sol·licitat; en els sistemes amb les interrupcions vectoritzades aquest valor especifica un índex dins la taula de vectors d'interrupció gestionada pel sistema operatiu.

L'execució d'una instrucció de programari duu a terme dues operacions:

1) Guardar el valor del registre d'estat i del PC a la pila del sistema perquè quan finalitzi l'execució de la RSI pugui retornar al punt d'execució on es trobava i continuar la seqüència d'execució.

2) Carregar en el PC l'adreça on es troba el codi de la RSI. Si és un sistema amb les interrupcions vectoritzades, l'adreça l'obtindrem de la taula de vectors d'interrupció.

Exemple d'instrucció d'interrupció de programari a l'arquitectura Intel x86-64

Instrucció	Operació	Exemple
INT servei	Crida al sistema	INT 80h

Exemple d'instrucció d'interrupció de programari a l'arquitectura CISCA

Instrucció	Operació	Exemple
INT servei	Crida al sistema	INT 1h

Respecte el **retorn d'una rutina de servei d'interrupció** s'ha de tenir en compte que abans de finalitzar l'execució d'una rutina de servei d'interrupció (RSI), hem de restaurar l'estat del processador per a tornar el control al programa que s'estava executant, per això cal recuperar els registres d'estat i el PC guardats a la pila del sistema i poder reprendre l'execució del programa que havíem aturat.

Exemple d'instruccions de crida i retorn de subrutina comunes a l'arquitectura Intel x86-64 i CISCA

Instrucció	Operació	Exemple
IRET	Retorn d'una rutina de servei d'interrupció	IRET

1.6.6. Instruccions d'entrada/sortida

Les instruccions d'entrada/sortida permeten llegir i escriure en un port d'E/S. Els ports d'E/S s'utilitzen per a accedir a un registre del mòdul d'E/S que controla un dispositiu o més, per a consultar-ne l'estat o programar-lo.

Segons l'arquitectura del computador podem tenir un mapa comú de memòria i E/S o un mapa independent d'E/S:

- **Mapa comú de memòria i E/S**: s'utilitzen les mateixes instruccions de transferència de dades explicades anteriorment.

Exemples de mapa comú

LOAD, STORE i MOV són instruccions que s'utilitzen al mapa comú de memòria i E/S.

- **Mapa independent d'E/S**: les instruccions utilitzades per a accedir als ports d'E/S són diferents de les de transferència de dades. Les més habituals són:
  - **IN**. Permet llegir d'un port d'E/S. Utilitza dos operands: un número de port i una localització per a emmagatzemar el valor llegit (normalment un registre).

– **OUT**. Permet escriure en un port d'E/S. Utilitza dos operands: un número de port i la localització del valor que s'ha d'escriure en el port, habitualment aquest valor es troba en un registre.

Exemple d'instruccions d'entrada/sortida en l'arquitectura Intel x86-64

Instrucció	Operació	Exemple
IN destinació, font	destinació ← port d'E/S indicat per l'operand font	IN AX, 20h
OUT destinació, font	port d'E/S indicat per l'operand destinació ← font	OUT 20h, EAX

Exemple d'instruccions d'entrada/sortida en l'arquitectura CISCA

Instrucció	Operació	Exemple
IN destinació, font	destinació ← port d'E/S indicat per l'operand font	IN R5, 01h
OUT destinació, font	port d'E/S indicat per l'operand destinació ← font	OUT 02h, R8

1.6.7. Altres tipus d'instruccions

En un joc d'instruccions hi sol haver altres tipus d'instruccions de caràcter més específic; algunes poden estar restringides al sistema operatiu per a ser executat en mode privilegiat.

Les instruccions de control del sistema, normalment, són instruccions privilegiades que en algunes màquines es poden executar només quan el processador es troba en estat privilegiat, com per exemple HALT (atura l'execució del programa), WAIT (espera per algun esdeveniment per a sincronitzar el processador amb altres unitats) i d'altres per a gestionar el sistema de memòria virtual o accedir a registres de control no visibles al programador.

Instruccions específiques per a treballar amb nombres en punt flotant (joc d'instruccions de la FPU) o per qüestions multimèdia, moltes de tipus SIMD (*single instruction multiple data*).

O d'altres com, per exemple, la instrucció NOP (no operació) que no fa res. Tot i que el ventall pot ser molt ampli.

1.7. Disseny del joc d'instruccions

El joc d'instruccions d'una arquitectura és l'eina que té el programador per a controlar el computador, per tant hauria de facilitar i simplificar la tasca del programador. D'altra banda, les característiques del joc d'instruccions condicionen el funcionament del processador i, per tant, tenen un efecte significatiu en el disseny del processador. Així, ens trobem amb el problema que moltes de les característiques que faciliten la tasca al programador dificulten el disseny del processador; per aquest motiu cal arribar a un compromís entre facilitar

el disseny del computador i satisfer les necessitats del programador. Una manera d'assolir aquest compromís és dissenyar un joc d'instruccions seguint un criteri d'ortogonalitat.

L'**ortogonalitat** consisteix en què totes les instruccions permetin utilitzar com a operand qualsevol dels tipus de dades existents i qualsevol dels modes d'adreçament, i en el fet que la informació del codi d'operació es limiti a l'operació que ha de fer la instrucció.

Si el joc d'instruccions és ortogonal serà regular i no presentarà casos especials, cosa que facilitarà la tasca del programador i la construcció de compiladors.

Els aspectes més importants que cal tenir en compte per a dissenyar un joc d'instruccions són els següents:

- **Conjunt d'operacions**: identificació de les operacions que cal dur a terme i la seva complexitat.
- **Tipus de dades**: identificació dels tipus de dades necessaris per a dur a terme les operacions.
- **Registres**: identificació del nombre de registres del processador.
- **Adreçament**: identificació dels modes d'adreçament que es poden utilitzar en els operands.
- **Format de les instruccions**: longitud i nombre d'operands, i mida dels diferents camps.

Disseny d'un joc d'instruccions

A causa de la forta interrelació entre els diversos aspectes que cal tenir presents per a dissenyar un joc d'instruccions, dissenyar un joc d'instruccions es converteix en una tasca molt complexa i no és l'objectiu d'aquesta assignatura abordar aquesta problemàtica.

2. Modes d'adreçament

Els operands d'una instrucció poden expressar directament una dada, l'adreça, o la referència a l'adreça, on tenim la dada. Aquesta adreça pot ser la d'un registre o la d'una posició de memòria, i en aquest últim cas l'anomenarem *adreça efectiva*.

Entenem per **mode d'adreçament** les maneres diferents d'expressar un operand en una instrucció i el procediment associat que permet obtenir l'adreça on està emmagatzemada la dada i, com a conseqüència, la dada.

Memòria virtual

Si tenim un sistema amb memòria virtual, l'adreça efectiva serà una adreça virtual i la correspondència amb l'adreça física dependrà del sistema de paginació i serà transparent al programador.

Els jocs d'instruccions ofereixen maneres diferents d'expressar els operands per mitjà dels seus modes d'adreçament, que seran un compromís entre diferents factors:

- El rang de les adreces que hem d'assolir per a poder accedir a tot l'espai de memòria adreçable pel que fa a programació.
- Pel que fa a l'espai per a expressar l'operand, s'intentarà reduir el nombre de bits que cal utilitzar per a codificar l'operand i, en general, les diferents parts de la instrucció per a poder fer programes que ocupin menys memòria.
- Les estructures de dades a què es pretengui facilitar l'accés o la manera d'operar-hi, com ara llistes, vectors, taules, matrius o cues.
- La complexitat de càlcul de l'adreça efectiva que expressa l'operand per a agilitar l'execució de les instruccions.

En aquest apartat analitzarem els modes d'adreçament més comuns, que es recullen en l'esquema següent:

Classificació de modes d'adreçament

La classificació de modes d'adreçament que presentem aquí s'ha fet a partir dels més utilitzats en els jocs d'instruccions de màquines reals, però en molts casos aquestes utilitzen variants dels modes d'adreçament explicats o els expressen de manera diferent.

Cal consultar el manual de referència del joc d'instruccions de cada màquina per a saber quins modes d'adreçament es poden utilitzar en cada operand de cada instrucció i la seva sintaxi.

- Adreçament immediat
- Adreçament directe:
  - Adreçament directe a registre
  - Adreçament directe a memòria
- Adreçament indirecte:
  - Adreçament indirecte a registre
  - Adreçament indirecte a memòria
- Adreçament relatiu:
  - Adreçament relatiu a registre base o relatiu
  - Adreçament relatiu a registre índex (RI) o indexat

  - Adreçament relatiu a PC

- Adreçament implícit
  - Adreçament a pila (indirecte a registre SP)

Cal tenir present que cada operand de la instrucció pot tenir el seu propi mode d'adreçament, i no tots els modes d'adreçament de què disposa un joc d'instruccions es poden utilitzar en tots els operands ni en totes les instruccions.

Hi ha una qüestió que, tot i ser transparent al programador, cal conèixer i tenir present perquè indirectament sí que el pot afectar a l'hora d'accedir a la memòria. Es tracta de l'ordenació dels bytes d'una dada quan aquesta té una mida superior a la mida de la paraula de memòria.

En la majoria de computadors la memòria s'adreça en bytes, és a dir, la mida de la paraula de memòria és d'un byte. Quan treballem amb una dada formada per diversos bytes caldrà decidir com s'emmagatzema la dada dins la memòria, és a dir, quin byte de la dada s'emmagatzema en cada posició de la memòria.

Es poden utilitzar dos sistemes diferents:

- **little-endian**: emmagatzemar el byte de menys pes de la dada a l'adreça de memòria més baixa.

- **big-endian**: emmagatzemar el byte de més pes de la dada a l'adreça de memòria més baixa.

Un cop escollit un d'aquests sistemes caldrà tenir-lo present i utilitzar-lo en tots els accessos a memòria (lectures i escriptures) per a assegurar la coherència de les dades.

Exemple

Suposem que volem emmagatzemar el valor hexadecimal següent: 12345678h. Es tracta d'un valor de 32 bits, format pels 4 bytes 12h 34h 56h i 78h. Suposem també que es vol emmagatzemar a la memòria a partir de l'adreça 200h. Com que cada posició de la memòria permet emmagatzemar un sol byte, necessitarem 4 posicions de memòria, corresponents a les adreces 200h, 201h, 202h i 203h.

Little-endian		**Big-endian**	
Adreça	**Contingut**	**Adreça**	**Contingut**
200h	78h	200h	12h
201h	56h	201h	34h
202h	34h	202h	56h
203h	12h	203h	78h

2.1. Adreçament immediat

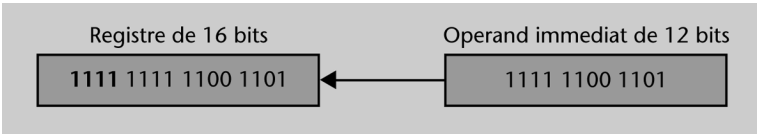
En l'adreçament immediat, l'operand expressa el valor de la dada que es vol utilitzar; és a dir, la dada és dins de la instrucció i el seu valor és fix.

Aquest mode d'adreçament s'acostuma a utilitzar en operacions aritmètiques o lògiques, transferències en les quals es vol inicialitzar registres i, de manera general, per a definir i fer servir constants.

El valor de la dada es representa, normalment, en complement a 2 i quan es transfereix a un registre o a una posició de memòria es fa l'extensió de signe replicant el bit de signe a l'esquerra fins a omplir l'operand destinació.

Exemple

Tenim un operand immediat de 12 bits en complement a 2 i volem transferir el nombre -51_{10} a un registre de 16 bits.



L'avantatge principal d'aquest mode d'adreçament és que no cal cap accés addicional a memòria per a obtenir la dada, cosa que agilita l'execució de la instrucció.

Els desavantatges principals són que el valor de la dada és constant i el rang de valors que es poden representar està limitat per la mida d'aquest operand, que no acostuma a ser massa gran, $[-2^{(n-1)}, 2^{(n-1)} - 1]$ si es representa en complement a 2, en què n és el nombre de bits de l'operand.

Instruccions de salt incondicional

En les instruccions de salt incondicional, com veurem més endavant, l'operand pot expressar l'adreça on es vol saltar; en aquest cas, aquesta adreça és la dada, ja que la funció d'aquesta instrucció és carregar aquest valor al PC i no cal fer cap accés a la memòria per a obtenir-la. Per aquest motiu, es considera un mode d'adreçament immediat, encara que l'operand expressi una adreça com en l'adreçament directe a memòria.

2.2. Adreçament directe

En l'adreçament directe l'operand indica on es troba la dada que es vol utilitzar. Si fa referència a un registre de la màquina, la dada estarà emmagatzemada en aquest registre i parlarem d'**adreçament directe a registre**; si fa referència a una posició de memòria, la dada estarà emmagatzemada en aquesta adreça de memòria (adreça efectiva) i parlarem d'**adreçament directe a memòria**.

Terminologia

Podeu trobar els modes d'adreçament directes referenciats amb altres noms:

- L'adreçament directe a registre com a directe absolut a registre o, simplement, a registre.
- L'adreçament directe a memòria com a absolut o directe absolut.

Exemple d'adreçament a registre i a memòria en l'arquitectura CISCA

Operands de 32 bits en complement a 2 i utilitzant format *little-endian*

	Codi oper.	Destinació	Font
Instrucció	MOV	R2	[0AB0 0100h]
Funció	R2 ← [0AB0 0100h] => R2 ← 01234567h		

Memòria		
Adreça	**Abans**	**Després**
0000 0000h		
...		
0AB0 0100h	67h	67h
0AB0 0101h	45h	45h
0AB0 0102h	23h	23h
0AB0 0103h	01h	01h
...		
0AB0 0200h		
0AB0 0201h		
0AB0 0202h		
0AB0 0203h		
...		
FFFF FFFFh		

Registres		
Registre	**Abans**	**Després**
R0		
R1		
R2	0000 0024h	0123 4567h
R3		
R4		
R5		
...		
R12		
R13		
R14		
R15/SP		
PC	0000 0100h	0000 0107h

Amb l'execució d'aquesta instrucció volem transferir el contingut de l'adreça de memòria 0AB00100h al registre R2; l'operand font fa referència a l'adreça 0AB00100h on s'ha de llegir la dada 01234567h. Abans de l'execució de la instrucció, el registre R2 conté el valor 00000024h i després el valor 01234567h, que és la dada que teníem a l'adreça de memòria 0AB00100h. A l'operand destinació tindrem un adreçament directe a registre, i a l'operand font, un adreçament directe a memòria.

Aquests modes d'adreçament tenen una forma molt simple i no cal fer càlculs per a obtenir l'adreça efectiva on hi ha la dada.

La mida de l'operand, en el cas de l'adreçament directe a registre dependrà del nombre de registres que tingui la màquina, que acostuma ser relativament reduït i, per tant, calen pocs bits; en el cas de l'adreçament directe a memòria

dependrà de la mida de la memòria. En les màquines actuals, l'operand haurà de ser molt gran per a poder adreçar tota la memòria, cosa que representa un dels inconvenients principals d'aquest mode d'adreçament.

Exemple

En una màquina amb 32 registres i 4 Mbytes de memòria ($4 \cdot 2^{20}$ bytes), necessitarem 5 bits per a codificar els 32 registres ($32 = 2^5$) i 22 bits per a codificar $4 \cdot 2^{20}$ adreces de memòria de 1 byte ($4\ M = 2^{22}$) i s'expressaran en binari pur, sense consideracions de signe.

Els avantatges principals de l'adreçament directe a registre és que la mida de l'operand és molt petita i l'accés als registres és molt ràpid, per la qual cosa és un dels modes d'adreçament més utilitzat.

2.3. Adreçament indirecte

En l'adreçament indirecte, l'operand indica on està emmagatzemada l'adreça de memòria (adreça efectiva) que conté la dada que volem utilitzar. Si fa referència a un registre de la màquina, l'adreça de memòria (adreça efectiva) que conté la dada estarà en aquest registre i parlarem d'**adreçament indirecte a registre**; si fa referència a una posició de memòria, l'adreça de memòria (adreça efectiva) que conté la dada estarà emmagatzemada en aquesta posició de memòria i parlarem d'**adreçament indirecte a memòria**.

Ja hem comentat que un dels problemes de l'adreçament directe a memòria és que calen adreces molt grans per a poder accedir a tota la memòria, amb el mode d'adreçament indirecte això no passa: es pot guardar tota l'adreça en un registre o en la memòria utilitzant les posicions que calguin.

Si es guarden en registres, pot passar que no tots els registres del processador es puguin utilitzar per a emmagatzemar aquestes adreces, però en aquest cas sempre n'hi haurà alguns d'especialitzats per a poder-ho fer, ja que són imprescindibles per a què el processador funcioni.

Generalment, les adreces que s'expressen en el mode d'adreçament indirecte a memòria són de mida inferior a les adreces reals de memòria, per aquest motiu només es podrà accedir a un bloc de la memòria, que habitualment es reserva com a taula de punters a les estructures de dades que utilitza el programa.

Aquest conveni serà el mateix per a cada màquina i sempre amb el mateix valor o la mateixa operació; per aquest motiu només es pot accedir a un bloc de la memòria adreçable del programa.

Exemple d'adreçament indirecte a registre en l'arquitectura CISCA

Operands de 32 bits en complement a 2 i utilitzant format *little-endian*

	Codi oper.	Destinació	Font
Instrucció	MOV	[R1]	R12
Funció	[0AB00100h] ← 01234567h		

Memòria		
Adreça	**Abans**	**Després**
0000 0000h		
…		
0AB0 0100h	00h	67h
0AB0 0101h	00h	45h
0AB0 0102h	00h	23h
0AB0 0103h	00h	01h
…		
0AB0 0200h		
0AB0 0201h		
0AB0 0202h		
0AB0 0203h		
…		
FFFF FFFFh		

Registres		
Registre	**Abans**	**Després**
R0		
R1	0AB0 0000h	0AB0 0100h
R2		
R3		
R4		
R5		
…		
R12	0123 4567h	0123 4567h
R13		
R14		
R15/SP		
PC	0000 0100h	0000 0107h

Amb l'execució d'aquesta instrucció es vol transferir el contingut del registre R12 a l'adreça de memòria 0AB00100h. L'operand destinació fa referència al registre R1 que conté l'adreça 0AB00100h on s'ha de guardar la dada, l'operand font fa referència al registre R12 on s'ha de llegir la dada 01234567h. En l'operand destinació tindrem un adreçament indirecte a registre, i en l'operand font, un adreçament directe a registre.

Exemple d'adreçament indirecte a memòria utilitzant el format de l'arquitectura CISCA

Tingueu present que aquesta instrucció segueix el format de l'arquitectura CISCA, però no forma part del seu joc d'instruccions perquè el mode d'adreçament indirecte a memòria no s'ha definit.

Operands de 32 bits en complement a 2 i utilitzant format *little-endian*

	Codi oper.	Destinació	Font
Instrucció	MOV	[[0AB00100h]]	R12
Funció	[0AB00200h] ← 01234567h		

Memòria		
Adreça	**Abans**	**Després**
0000 0000h		
...		
0AB0 0100h	00h	00h
0AB0 0101h	02h	02h
0AB0 0102h	B0h	B0h
0AB0 0103h	0Ah	0Ah
...		
0AB0 0200h	00h	67h
0AB0 0201h	00h	45h
0AB0 0202h	00h	23h
0AB0 0203h	00h	01h
...		
FFFF FFFFh		

Registres		
Registre	**Abans**	**Després**
R0		
R1		
R2		
R3		
R4		
R5		
...		
R12	0123 4567h	0123 4567h
R13		
R14		
R15/SP		
PC	0000 0100h	0000 0107h

Amb l'execució d'aquesta instrucció es vol transferir el contingut del registre R12 a l'adreça de memòria 0AB00200h. L'operand destinació fa referència a l'adreça 0AB00100h; en aquesta adreça hi ha emmagatzemada l'adreça 0AB00200h, que és on s'ha de guardar la dada, l'operand font fa referència al registre R12 on s'ha de llegir la dada 01234567h. En l'operand destinació tindrem un adreçament indirecte a memòria, i en l'operand font, un adreçament directe a registre.

El desavantatge principal d'aquest mode d'adreçament és que necessita un accés més a memòria que el directe. És a dir, un accés a memòria per l'adreçament indirecte a registre i dos accessos a memòria per l'adreçament indirecte a memòria; per aquest motiu aquest segon mode d'adreçament no s'implementa en la majoria de màquines.

2.4. Adreçament relatiu

En l'adreçament relatiu, l'operand expressarà dos valors, una adreça de memòria i un desplaçament respecte d'aquesta adreça (llevat dels casos en què un dels dos sigui implícit). L'adreça de memòria (adreça efectiva) on tindrem la dada l'obtindrem sumant el desplaçament a l'adreça de memòria. Les variants més utilitzades d'aquest mode d'adreçament són **adreçament relatiu a registre base**, **adreçament relatiu a registre índex** i **adreçament relatiu a PC**.

Terminologia

Podeu trobar l'adreçament relatiu, i les seves variants, referenciat com a *adreçament amb desplaçament* o *adreçament indexat*.

Aquests modes d'adreçament són molt potents, no requereixen accessos extres a la memòria com passa amb l'indirecte, però cal fer una suma, que no retarda gaire l'execució de la instrucció, especialment en les màquines que tenen una unitat específica per al càlcul d'aquestes adreces.

Des del punt de vista del programador, aquests modes d'adreçament són molt útils per a accedir a estructures de dades com vectors, matrius o llistes, ja que s'aprofita la localitat de les dades, atès que la majoria de referències en la memòria són molt properes entre si.

Aquests modes d'adreçament també són la base per a fer que els codis siguin reentrants i reubicables, ja que permeten canviar les referències a les adreces de memòria canviant simplement el valor d'un registre, per a accedir tant a les dades com al codi mitjançant les adreces de les instruccions de salt o crides a subrutines; també són útils en algunes tècniques per a protegir espais de memòria i per a implementar-ne la segmentació. Aquests usos s'aprofiten en la compilació i en la gestió que fa el sistema operatiu dels programes en execució i, per tant, són transparents al programador.

2.4.1. Adreçament relatiu a registre base

En l'adreçament relatiu a registre base, l'adreça de memòria s'emmagatzema en el registre que anomenarem *registre base* (RB) i el desplaçament es troba explícitament en la instrucció. És més compacte que el mode d'adreçament absolut a memòria ja que el nombre de bits utilitzats per al desplaçament és inferior al nombre de bits d'una adreça de memòria.

En algunes instruccions el registre base s'utilitza de manera implícita i, per tant, només caldrà especificar explícitament el desplaçament.

Aquest mode d'adreçament s'utilitza sovint per a implementar la segmentació. Algunes màquines tenen registres específics per a això i s'utilitzen de manera implícita, mentre que en altres màquines es poden triar els registres que es volen utilitzar com a base del segment, però, aleshores, cal especificar-ho explícitament en la instrucció.

Segmentació dels Intel x86-64

En el cas de l'Intel x86-64, s'implementa la segmentació utilitzant registres de segment específics (CS:codi, DS:dades, SS:pila i ES, FS, GS:extra), però també es té la flexibilitat de poder reassignar-ne alguns, tant si és de manera explícita a les instruccions o mitjançant directives de compilació com l'ASSUME.

2.4.2. Adreçament relatiu a registre índex

En l'adreçament relatiu a registre índex, l'adreça de memòria es troba explícitament en la instrucció i el desplaçament s'emmagatzema en el registre que anomenarem *registre índex* (RI). Just al contrari que en l'adreçament relatiu a registre base.

Aquest mode d'adreçament s'utilitza sovint per a accedir a estructures de dades com ara vectors, matrius o llistes, per la qual cosa és molt habitual que després de cada accés s'incrementi o decrementi el registre índex un valor constant (que depèn de la mida i el nombre de posicions de memòria de les dades a les quals s'accedeix). Per aquest motiu, algunes màquines fan aquesta operació de manera automàtica i proporcionen diferents alternatives que denominarem *adreçaments relatius autoindexats*. Si l'autoindexació es fa sobre registres de caràcter general pot requerir un bit extra per a indicar-ho en la codificació de la instrucció.

És molt habitual permetre tant l'autoincrement com l'autodecrement, operació que es pot fer abans d'obtenir l'adreça efectiva o després. Per tant, hi haurà quatre alternatives d'implementació, tot i que un mateix joc d'instruccions no les tindrà simultàniament:

1) Preautoincrement: RI s'incrementa abans d'obtenir l'adreça.

2) Preautodecrement: RI es decrementa abans d'obtenir l'adreça.

3) Postautoincrement: RI s'incrementa després d'obtenir l'adreça.

4) Postautodecrement: RI es decrementa després d'obtenir l'adreça.

Exemple d'adreçament relatiu a registre base en l'arquitectura CISCA

Operands de 32 bits en complement a 2 i utilitzant format *little-endian*

	Codi oper.	Destinació	Font
Instrucció	MOV	[R2+0100h]	R12
Funció	[0AB0 0000h+0100h] ← 01234567h		

Memòria		
Adreça	**Abans**	**Després**
0000 0000h		
...		
0AB0 0100h	00h	67h
0AB0 0101h	00h	45h
0AB0 0102h	00h	23h
0AB0 0103h	00h	01h
...		
0AB0 0200h		
0AB0 0201h		
0AB0 0202h		
0AB0 0203h		
...		
FFFF FFFFh		

Registres		
Registre	**Abans**	**Després**
R0		
R1		
R2	0AB0 0000h	0AB0 0000h
R3		
R4		
R5		
...		
R12	0123 4567h	0123 4567h
R13		
R14		
R15/SP		
PC	0000 0100h	0000 0105h

Amb l'execució d'aquesta instrucció es vol transferir el contingut del registre R12 a l'adreça de memòria 0AB00100h. L'operand destinació fa referència al registre R2, que fa de registre base i conté l'adreça 0AB00000h i al desplaçament 0100h, que, sumat a l'adreça que hi ha a R2, dóna l'adreça de memòria on s'ha de guardar la dada (posició 0AB00100h); l'operand font fa referència al registre R12 on s'ha de llegir la dada 01234567h. En l'operand destinació tindrem un adreçament relatiu a registre base, i en l'operand font, un adreçament directe a registre.

Exemple d'adreçament relatiu a registre índex en l'arquitectura CISCA

Operands de 32 bits en complement a 2 i utilitzant format *little-endian*

	Codi oper.	Destinació	Font
Instrucció	MOV	[0AB00000h+R1]	R12
Funció	[0AB0 0000h+0200h] ← 01234567h		

Memòria		
Adreça	**Abans**	**Després**
0000 0000h		
...		
0AB0 0100h		
0AB0 0101h		
0AB0 0102h		
0AB0 0103h		
...		
0AB0 0200h	00h	67h
0AB0 0201h	00h	45h
0AB0 0202h	00h	23h
0AB0 0203h	00h	01h
...		
FFFF FFFFh		

Registres		
Registre	**Abans**	**Després**
R0		
R1	0000 0200h	0000 0200h
R2		
R3		
R4		
R5		
...		
R12	0123 4567h	0123 4567h
R13		
R14		
R15/SP		
PC	0000 0100h	0000 0107h

Amb l'execució d'aquesta instrucció es vol transferir el contingut del registre R12 a l'adreça de memòria 0AB00200h. L'operand destinació fa referència a l'adreça 0AB00000h i al registre R1, que fa de registre índex i conté el desplaçament 00000200h, que, sumats, donen l'adreça de memòria on s'ha de guardar la dada (posició 0AB00200h); l'operand font fa referència al registre R12 on s'ha de llegir la dada 01234567h. En l'operand destinació tindrem un adreçament relatiu a registre índex, i en l'operand font, un adreçament directe a registre.

Moltes màquines habitualment implementen variants d'aquests modes d'adreçament, combinant l'adreçament relatiu a registre base i l'adreçament relatiu a registre índex. Utilitzen dos registres o més i el desplaçament, cosa que evita posar-hi l'adreça de memòria. Tenen tant o més potència d'accés i es pot reduir força la mida de la instrucció, ja que només cal identificar registres o registres i un desplaçament.

2.4.3. Adreçament relatiu a PC

L'adreçament relatiu a PC és equivalent al relatiu a registre base, amb la diferència que utilitza el registre comptador de programa (PC) de manera implícita en la instrucció i només cal expressar, mitjançant una etiqueta, el desplaçament per a calcular l'adreça de memòria (adreça efectiva) a la qual es vol accedir.

Una altra característica que diferencia el mode d'adreçament relatiu a PC és la manera d'expressar el desplaçament: com que pot ser un valor tant positiu com negatiu, s'acostuma a representar en complement a 2.

L'adreçament relatiu a PC s'utilitza sovint en les instruccions de ruptura de seqüència, generalment en els salts condicionals, que acostumen a ser curts i, en conseqüència, la mida del camp no haurà de ser gaire gran.

Exemple d'adreçament relatiu a registre PC en l'arquitectura CISCA

Tenim el fragment de codi següent:

Memòria		
Adreça	**Etiqueta**	**Instrucció**
0001 1000h		MOV R0,0
0001 1007h	bucle:	ADD R0,1
0001 100Eh		CMP R0,8
0001 1015h		JE bucle
0001 1019h		MOV R0,-1

En aquest codi, cada instrucció ocupa 7 bytes, llevat la instrucció JE bucle, que ocupa 4 bytes. L'etiqueta *bucle* fa referència a l'adreça de memòria 00001007h. La instrucció *JE bucle* (salta si el bit de zero està actiu) utilitza adreçament relatiu a PC i per tant no codifica l'adreça de memòria a la qual es vol saltar sinó el desplaçament respecte del PC actualitzat utilitzant 16 bits (desplaçament = etiqueta-$PC_{updated}$).

En el cicle d'execució de la instrucció, l'actualització del PC es fa a les primeres fases d'aquest cicle, per tant, al final de l'execució de la instrucció (que és quan sabrem si hem de saltar) el PC no apuntarà a la instrucció de salt (JE bucle; PC=00011015h), sinó a la instrucció següent (MOV R0,–1; $PC_{updated}$ = 00011019h); és a dir, haurem de saltar tres instruccions enrere i com que la instrucció JE bucle ocupa 4 bytes i la resta 7 bytes cadascuna, el desplaçament serà de –18; per a obtenir aquest valor restarem de l'adreça de l'etiqueta *bucle* el valor de $PC_{updated}$ (00011007h – 00011019h = FFEEh(–18 en Ca2)). Aquest valor, FFEEh, és el que es codificarà com a desplaçament a la instrucció de salt condicional.

Aleshores, quan s'executa una instrucció de salt condicional, si es compleix la condició s'actualitza el PC sumant-li el desplaçament que tenim codificat. En el nostre cas, després d'executar la instrucció de salt condicional *JE bucle* si el bit de zero Z està actiu, saltem i PC valdrà 00011007h i si el bit de zero Z no està actiu, no saltem i PC valdrà 00011019h.

El compilador fa la conversió de les etiquetes utilitzades en un programa a un desplaçament de manera transparent al programador perquè el mode d'adreçament és implícit en les instruccions de salt.

És molt important per al programador saber quin mode d'adreçament utilitza cada instrucció de ruptura de seqüència, perquè el desplaçament que es pot especificar en l'adreçament relatiu a PC no és gaire gran i pot passar que no permeti arribar a l'adreça desitjada; aleshores caldrà modificar el programa per a poder utilitzar una altra instrucció de ruptura de seqüència amb un adreçament que permeti arribar a tot l'espai adreçable.

Exemple

En l'exemple anterior s'utilitzen 16 bits per al desplaçament, per tant, es poden saltar 32.767 posicions endavant i 32.768 posicions enrere. Així doncs, el rang d'adreces a què es pot accedir des de l'adreça apuntada pel $PC_{updated}$ (00011019h), és des de l'adreça 00009019h a l'adreça 00019018h; si es vol saltar fora d'aquest espai no es pot utilitzar aquesta instrucció perquè utilitza adreçament relatiu a PC.

2.5. Adreçament implícit

En l'adreçament implícit, la instrucció no conté informació sobre la localització de l'operand perquè aquest es troba en un lloc predeterminat, i queda especificat de manera implícita en el codi d'operació.

2.5.1. Adreçament a pila

L'adreçament a la pila és un mode d'adreçament implícit; és a dir, no cal fer una referència explícita a la pila, sinó que treballa implícitament amb el cim de la pila per mitjà de registres de la màquina; habitualment un d'aquests registres s'anomena *stack pointer* (SP) i s'utilitza per a apuntar al cim de la pila.

Pila

Una pila és una llista d'elements amb l'accés restringit, de manera que només es pot accedir als elements d'un extrem de la llista. Aquest extrem s'anomena *cim de la pila*. L'últim element que entra a la pila és el primer que en surt (LIFO). A mesura que s'hi van afegint elements, la pila creix i segons la implementació ho farà cap a adreces més petites (mètode més habitual) o cap a adreces més grans.

Com que és un mode d'adreçament implícit, només s'utilitza en instruccions determinades, les més habituals de les quals són PUSH (posar un element a la pila) i POP (treure un element de la pila).

Aquest mode d'adreçament es podria entendre com un adreçament indirecte a registre afegint la funcionalitat d'autoindexat que ja hem comentat. És a dir, s'accedeix a una posició de memòria identificada per un registre, el registre SP, que s'actualitza, abans o després de l'accés a memòria, perquè apunti al nou cim de la pila.

Exemple

Suposem que cada element de la pila és una paraula de 2 bytes, la memòria s'adreça a nivell de byte i SP apunta a l'element que és al cim de la pila.

Per a posar un element en la pila (PUSH X) caldrà fer el següent:

Creix cap a adreces més petites (preautodecrement)	Creix cap a adreces més grans (preautoincrement)
SP = SP – 2 M[SP] = X	SP = SP + 2 M[SP] = X

Per a treure un element de la pila (POP X) caldrà fer el següent:

Creix cap a adreces més petites (postautoincrement)	Creix cap a adreces més grans (postautodecrement)
X = M[SP] SP = SP + 2	X = M[SP] SP = SP – 2

Exemple d'adreçament a pila en l'arquitectura CISCA

Operands de 32 bits en complement a 2 i utilitzant format *little-endian*. La pila creix cap a adreces petites

	Codi oper.	Font
Instrucció	PUSH	R2
Funció	[SP – 4] ← 01234567h	

Memòria		
Adreça	**Abans**	**Després**
0000 0000h		
...		
0AB0 0100h		
0AB0 0101h		
0AB0 0102h		
0AB0 0103h		
...		
FFFF FFFAh		
FFFF FFFBh		
FFFF FFFCh	00h	67h
FFFF FFFDh	00h	45h
FFFF FFFEh	00h	23h
FFFF FFFFh	00h	01h

Registres		
Registre	**Abans**	**Després**
R0		
R1		
R2	01234567h	01234567h
R3		
R4		
R5		
...		
R12		
R13		
R14		
R15/SP	0000 0000h	FFFF FFFCh

PC	0000 0100h	0000 0107h

Amb l'execució d'aquesta instrucció es vol transferir el contingut del registre R2 al cim de la pila. L'operand font fa referència al registre R2 on s'ha de llegir la dada 01234567h que volem posar a la pila. Per a guardar aquesta dada a la pila primer es decrementa en 4 el registre SP (en CISCA, el registre R15), perquè les dades són de 32 bits. El registre SP inicialment val 00000000h, 00000000h – 4 = FFFFFFFCh (–4 en Ca2). Després, de la

mateixa manera que en un adreçament indirecte a registre, utilitzarem aquesta adreça que tenim al registre SP per a emmagatzemar el valor que tenim a R2 a la memòria, i SP quedarà apuntant al cim de la pila. Si SP val 0 voldrà dir que la pila és buida.

Resum

Hem començat parlant de les característiques principals dels jocs d'instruccions dels quals hem destacat diferents punts.

Hem vist que el cicle d'execució de la instrucció es divideix en les quatre fases següents:

Fase 1: Lectura de la instrucció

Fase 2: Lectura dels operands font

Fase 3: Execució de la instrucció i emmagatzematge de l'operand destinació

Fase 4: Comprovació d'interrupcions

El tipus d'arquitectura del joc d'instruccions depèn de la localització dels operands i a partir d'aquí hem definit cinc tipus d'arquitectures: pila, acumulador, registre-registre, registre-memòria, memòria-memòria.

La representació del joc d'instruccions es fa des de dos punts de vista:

- El punt de vista del programador, que anomenem *llenguatge d'assemblador*.
- El punt de vista del computador, que anomenem *llenguatge de màquina*.

Pel que fa al format de les instruccions, hem vist el següent:

- Els elements que componen la instrucció: codi d'operació, operands font, operand destinació i adreça de la instrucció següent.
- La mida de les instruccions, que pot ser fixa o variable, i com determinar la mida dels diferents camps.

Pel que fa als operands de la instrucció hem analitzat el nombre d'operands que pot tenir i la seva localització, i el tipus i la mida de les dades que tractem amb els operands.

Els tipus d'instruccions vistos en el mòdul apareixen referenciats en la taula següent.

Tipus d'instruccions		Exemples
Instruccions de transferència de dades		MOV AL, 0

Tipus d'instruccions		Exemples
		MOV R1, R2
Instruccions aritmètiques	Suma	ADD R1, R2
	Resta	SUB R1, 2
	Multiplicació	MUL R1, R2
	Divisió	DIV R2, R3
	Increment	INC RAX
	Decrement	DEC RAX
	Comparació	CMP RAX, RBX
	Negació	NEG RAX
Instruccions lògiques	AND	AND R1, 1
	OR	OR R1, R2
	XOR	XOR R1, R2
	NOT	NOT R1
	Desplaçament lògic a l'esquerra	SHL RAX, 1
	Desplaçament lògic a la dreta	SHR RAX, 1
	Desplaçament aritmètic a l'esquerra	SAL RAX, 1
	Desplaçament aritmètic a la dreta	SAR RAX, 1
	Rotació a l'esquerra	ROL RAX, 1
	Rotació a la dreta	ROR RAX, 1
Instruccions de ruptura de seqüència	Salt incondicional	JMP etiqueta
	Salt condicional	JE etiqueta
	Instruccions de crida i retorn de subrutina	CALL etiqueta
	Instruccions d'interrupció de programari	INT 80h
	Instruccions de retorn d'interrupció	IRET
Instruccions d'entrada/sortida	Entrada	IN AL, 20h
	Sortida	OUT 20h, AL
Altres tipus d'instruccions		NOP

També s'han explicat detalladament els diferents modes d'adreçament que pot tenir un joc d'instruccions, que estan resumits en la taula següent.

Adreçament	Sintaxi	Que expressa OP	Com obtenim l'adreça	Com obtenim la dada	Observacions
Immediat	Valor	OP = Dada		Dada = OP	
Directe a registre	R	OP = R		Dada = R	
Directe a memòria	[A]	OP = A	AE = A	Dada = M[A]	
Indirecte a registre	[R]	OP = R	AE = R	Dada = M[R]	
Indirecte a memòria	[[A]]	OP = A	AE = M[A]	Dada = M[M[A]]	
Relatiu a RB	[RB + Desp.]	OP = RB + Desp.	AE = RB + Desp.	Dada = M[RB + Desp.]	
Relatiu a RI	[A + RI]	OP = A + RI	AE = A + RI	Dada = M[A + RI]	Amb preautoincrement o postautoincrement RI = RI ± 1
Relatiu a PC	A	OP = Desp.		Dada = PC + Desp.	PC és implícit i Dada és l'adreça de la instrucció següent per executar.
A pila	Valor o R o A	OP = Dada o R o A	AE = SP	Dada = M[SP]	Adreçament implícit

Abreviatures de la taula. Dada: amb la qual volem operar; OP: informació expressada a l'operand de la instrucció; A: adreça de memòria; AE: adreça efectiva de memòria (adreça on és la dada); R: referència d'un registre; Desp.: desplaçament; [ ]: per a indicar accés a memòria; M[A]: contingut d'una adreça de memòria; M[R]: contingut de l'adreça de memòria que hi ha al registre R.