Ejemplo: Máquina Sencilla

Curso 2007 - 2008
Asignatura: Laboratorio de Computadores
3º Ingeniería en Informática
Departamento de Informática e Ingeniería de Sistemas
Centro Politécnico Superior

Ejemplo: máquina sencilla

1.- Introducción.

Vamos a ver el microprograma correspondiente a una máquina sencilla diferente a la original ya que el código de operación de la instrucción ocupa 8 bits (los de mayor peso). Esto hará que sólo queden 8 bits para expresar las direcciones de los operandos fuente y destino en vez de los 14 que serían necesarios. Como no se puede hacer así vamos a ver otras posibilidades:

- de los ocho bits de mayor peso utilizar los de menor peso (bits 9-8) para el código de instrucción y los demás (bits 15-10) junto con los de menor peso para los operandos. Esto complicaría la búsqueda de operandos y este ejemplo quiere ser mucho más sencillo.

- la solución adoptada consistirá en que las direcciones de los operandos vengan en direcciones de RAM consecutivas a la que ocupa la instrucción; se pierde mucho espacio pero es muy sencillo (explicación gráfica del punto 4, apartado 4).

Supongamos que, entendido un determinado enunciado, ya tenemos pensado el microcódigo. Veamos un poco el proceso de manejo de aplicaciones y programas:

1.- Entramos en la aplicación *Copro II - MicroAss* y escribimos el código (programa maqsen del apartado 2). A continuación hemos de compilarlo con la opción *Ensamblador/Ensamblar*. Esto creará la versión **.mic** (código ejecutable que sólo creará si no hubo errores de compilación, apartado 7).

2.- Entrar en el simulador de COPRO ejecutando *Copro II - Monitor*, cargar en la micromemoria el **.mic** creado en el punto anterior (para ello opción *Cargar la Micromemoria* del menú *Archivo*, apartado 1.2 del Manual de Usuario).

3.- Escribir un pequeño programa en la macromemoria o una instrucción en concreto que se quiera probar. Se puede hacer directamente sobre la ventana, o escribir el programa (conforme al formato explicado en el apartado 2.8.1 del enunciado de la práctica) en algún fichero en el que se salvó previamente la macromemoria, y cargarlo desde dentro de COPRO con la opción Cargar la Macromemoria del menú Archivo (apartado 1.2 del Manual de Usuario).

4.- Inicializar registros, flags, etc, (apartado 1.2 del Manual de Usuario) y posicionar el secuenciador en la microinstrucción 0 con la opción *Saltar a 0*.

5.- Ejecución en cualquiera de sus formas (apartado 1.3 del Manual de Usuario).

6.- Entrada de puntos de paro en algún lugar concreto que queramos parar la ejecución (si no se ha hecho previamente en el microcódigo con la instrucción HALT) u otras modificaciones sobre la marcha. Desensamblaje en la ventana Decodificación de la microinstrucción modificada para estar seguros o por si no sabíamos la sintaxis necesaria en el microcódigo para hacer algo que sí sabemos hacer cambiando la micromemoria directamente.

Se puede desensamblar todo un microcódigo almacenado en la versión .mic utilizando la aplicación *Copro II - DisAss*, fuera del monitor; se obtendrá un fichero **.dis** como el del apartado 7. Lógicamente, este fichero será equivalente al original que contenía el microcódigo pues hemos ensamblado y desensamblado.

7.- Salvar la micromemoria y la macromemoria con *Guardar...* del menú Archivo.

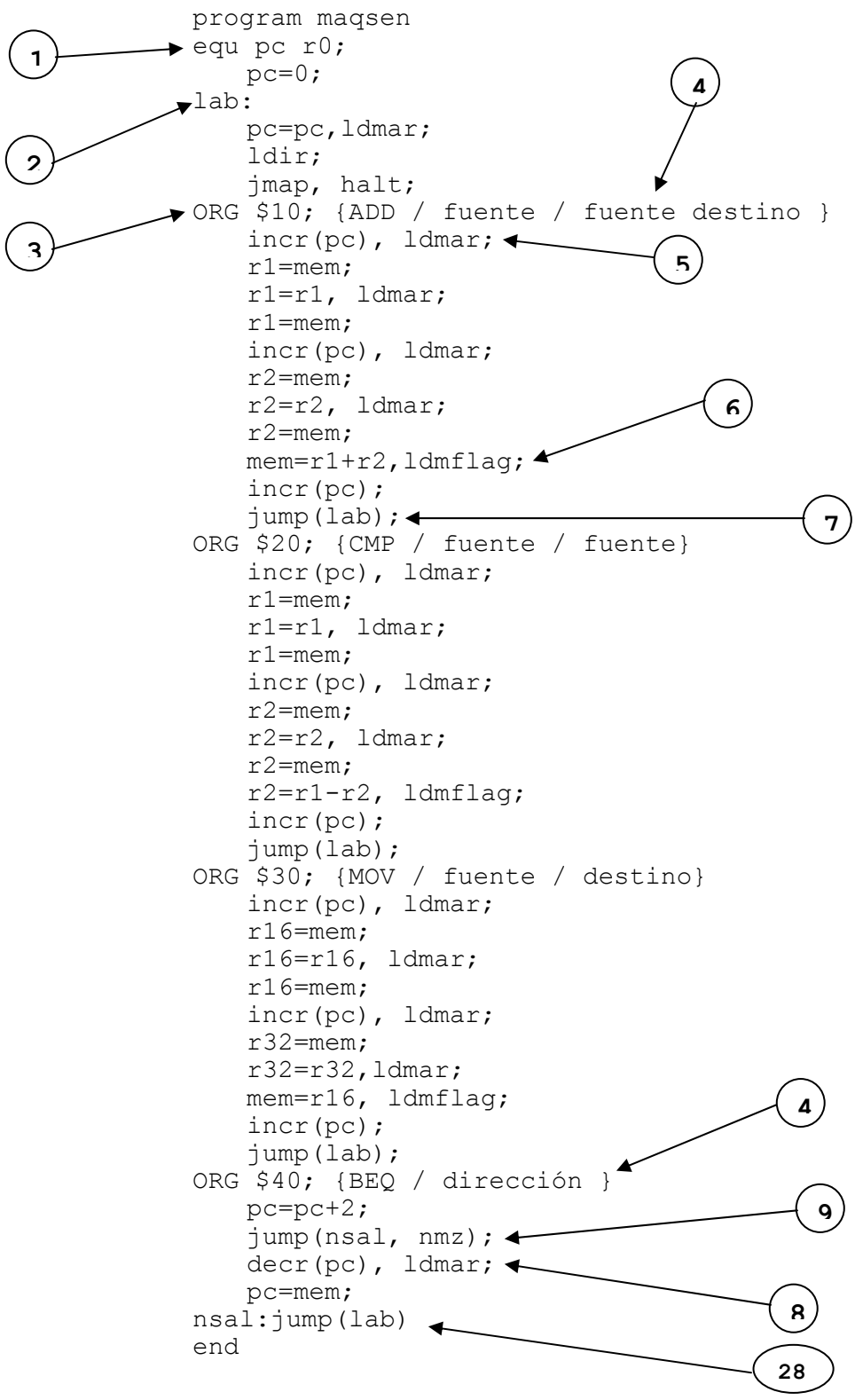
NOTA IMPORTANTE: a veces uno pretende hacer algo en el fichero del microcódigo pero no sabe la sintaxis y no encuentra nada en los ejemplos como es normal; puede hacer varias cosas:

- puede preguntarle al vecino que tal vez pasó ya por este problema (será el caso más fácil pero no siempre posible).
- puede modificar directamente la micromemoría dentro del monitor hasta conseguir lo que desea, una vez conseguido ver en la ventana Decodificación su traducción a microcódigo como hemos explicado antes.
- puede intentar encontrar su caso dentro del apdo Definición y sintaxis del microensamblador de COPRO (apdo 9 del Microensamblador).
- pero no puede preguntarle al profesor de la asignatura dichas pequeñeces, sólo cosas más interesantes como las que siguen.

En algún caso incluso puede ocurrir que utilizando la sintaxis que el desensamblaje nos indica, el compilador no la acepte. Puede ocurrir también que la definición de una microinstrucción con varias microoperaciones obedezca perfectamente a lo explicado en el apartado 8 del manual de microensamblador, y no funcione. Puede haber fallos en el compilador. En este caso habrá que reconstruir menos eficientemente la microinstrucción e incluso modificar a mano la micromemoria siempre que se vaya a usar.

2.- maqsen

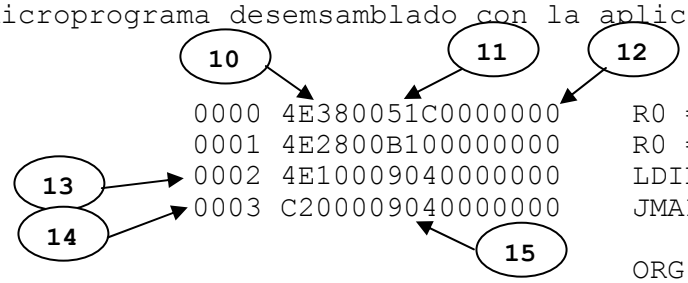
(microprograma fuente realizado con la aplicación MicroAss)



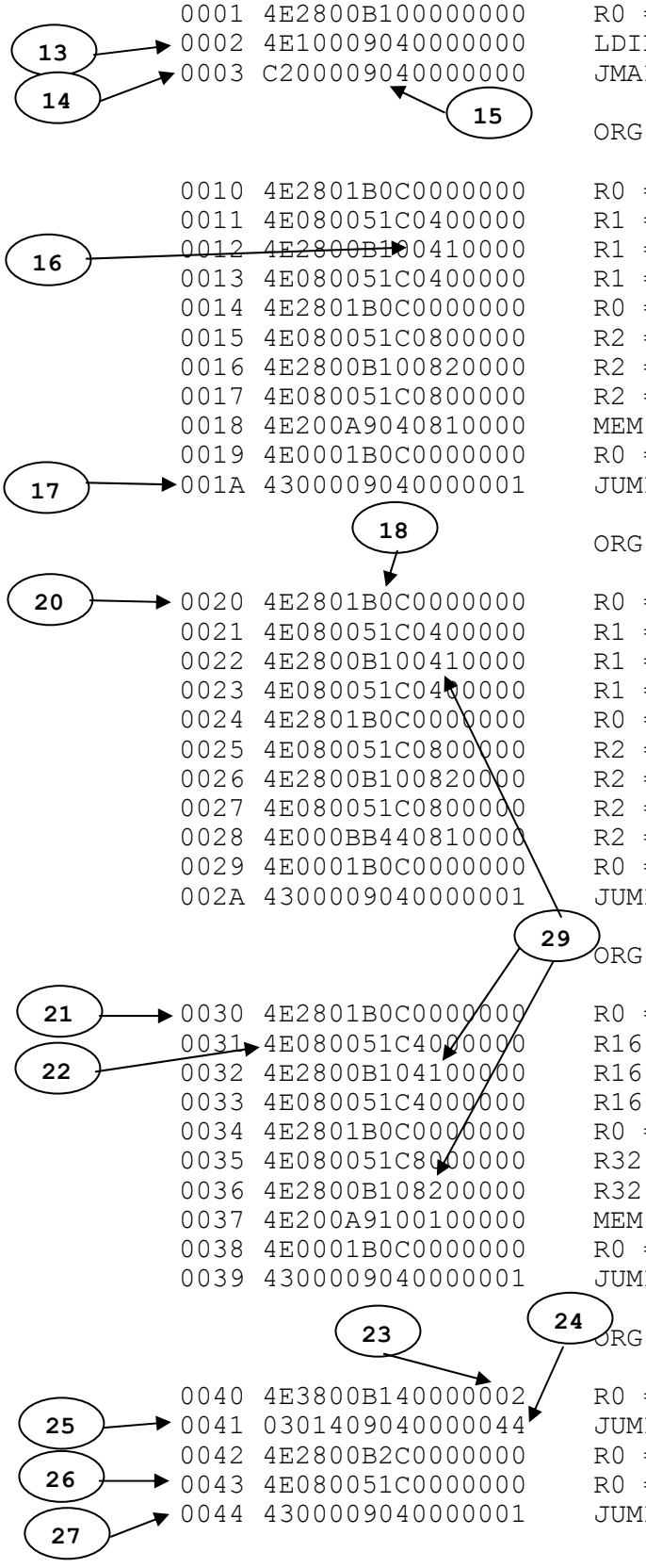

```
program maqsen
equ pc r0;
    pc=0;
lab:
    pc=pc,ldmar;
    ldir;
    jmap, halt;
ORG $10; {ADD / fuente / fuente destino }
    incr(pc), ldmar;
    r1=mem;
    r1=r1, ldmar;
    r1=mem;
    incr(pc), ldmar;
    r2=mem;
    r2=r2, ldmar;
    r2=mem;
    mem=r1+r2,ldmflag;
    incr(pc);
    jump(lab);
ORG $20; {CMP / fuente / fuente}
    incr(pc), ldmar;
    r1=mem;
    r1=r1, ldmar;
    r1=mem;
    incr(pc), ldmar;
    r2=mem;
    r2=r2, ldmar;
    r2=mem;
    r2=r1-r2, ldmflag;
    incr(pc);
    jump(lab);
ORG $30; {MOV / fuente / destino}
    incr(pc), ldmar;
    r16=mem;
    r16=r16, ldmar;
    r16=mem;
    incr(pc), ldmar;
    r32=mem;
    r32=r32,ldmar;
    mem=r16, ldmflag;
    incr(pc);
    jump(lab);
ORG $40; {BEQ / dirección }
    pc=pc+2;
    jump(nsal, nmz);
    decr(pc), ldmar;
    pc=mem;
nsal:jump(lab)
end
```

3.- maqsen.dis

(microprograma desemsamblado con la aplicación DisAss)




```
0000 4E380051C0000000    R0 = 0;
0001 4E2800B100000000    R0 = R0, LDMAR;
0002 4E10009040000000    LDIR;
0003 C200009040000000    JMAP, HALT;

                         ORG $0010;

0010 4E2801B0C0000000    R0 = R0 + 1, LDMAR;
0011 4E080051C0400000    R1 = MEM;
0012 4E2800B100410000    R1 = R1, LDMAR;
0013 4E080051C0400000    R1 = MEM;
0014 4E2801B0C0000000    R0 = R0 + 1, LDMAR;
0015 4E080051C0800000    R2 = MEM;
0016 4E2800B100820000    R2 = R2, LDMAR;
0017 4E080051C0800000    R2 = MEM;
0018 4E200A9040810000    MEM = R1 + R2, LDMFLAG;
0019 4E0001B0C0000000    R0 = R0 + 1;
001A 4300009040000001    JUMP($0001);

                         ORG $0020;

0020 4E2801B0C0000000    R0 = R0 + 1, LDMAR;
0021 4E080051C0400000    R1 = MEM;
0022 4E2800B100410000    R1 = R1, LDMAR;
0023 4E080051C0400000    R1 = MEM;
0024 4E2801B0C0000000    R0 = R0 + 1, LDMAR;
0025 4E080051C0800000    R2 = MEM;
0026 4E2800B100820000    R2 = R2, LDMAR;
0027 4E080051C0800000    R2 = MEM;
0028 4E000BB440810000    R2 = R1 - R2, LDMFLAG;
0029 4E0001B0C0000000    R0 = R0 + 1;
002A 4300009040000001    JUMP($0001);

                         ORG $0030;

0030 4E2801B0C0000000    R0 = R0 + 1, LDMAR;
0031 4E080051C4000000    R16 = MEM;
0032 4E2800B104100000    R16 = R16, LDMAR;
0033 4E080051C4000000    R16 = MEM;
0034 4E2801B0C0000000    R0 = R0 + 1, LDMAR;
0035 4E080051C8000000    R32 = MEM;
0036 4E2800B108200000    R32 = R32, LDMAR;
0037 4E200A9100100000    MEM = R16, LDMFLAG;
0038 4E0001B0C0000000    R0 = R0 + 1;
0039 4300009040000001    JUMP($0001);

                         ORG $0040;

0040 4E3800B140000002    R0 = 2 + R0;
0041 0301409040000044    JUMP($0044, NMZ);
0042 4E2800B2C0000000    R0 = R0 - 1, LDMAR;
0043 4E080051C0000000    R0 = MEM;
0044 4300009040000001    JUMP($0001);
```



4.- explicaciones detalladas.

El.prograrna del microcódigo empieza siempre con program <*nombre de programa*>. A continuación se definen los indentificadores **equ**, asignaciones iniciales de registros, y luego la búsqueda de la instrucción a donde volveremos después de la ejecución del grupo de microinstrucciones correspondientes a cada instrucción:

lab: pc=pc, ldmar; carga del registro mar con el valor del pc.
Idir; lectura de memoria y carga del IR.
jmap; dirección de salto en el secuenciador para seguir ejecutando en la microdirección donde se encuentra el microcódigo correspondiente a la instrucción que se acaba de cargar en IR (dirección acompañada por ORG y que se forma según la idea explicada en los apartados 2.1 y 3.2 del capítulo Arquitectura de COPRO).

Pasemos a detallar los puntos de los ficheros maqsen (fuente) y maqsen.dis:

1.- **equ pc r0**; identificador, de tal forma que para todo el programa nos podremos referir a *pc* y no a *r0*.
2.- **label:** etiqueta a la que nos podemos referir en los saltos.
3.- **ORG $10;** en la dirección 10 (en hexa) empieza el código referente a la instrucción de código de instrucción 01. No habrá, por tanto, instrucción con código de instrucción 00.
4.- Disposición en la macromernoria de la instrucción ADD y las direcciones de los operandos fuente y destino como hemos explicado en la introducción de este capítulo. Instrucción BEQ en la macromemoria y dirección de salto en posiciones consecutivas.

	8	4	4
@ i	ADD	X	X
@ i+1	@ fuente		
@ i+2	@ destino		

	8	4	4
@ i	BEQ	X	X
@ i+1	@ destino del salto		

5.- **incr(pc), ldmar;** microinstrucción con dos microoperaciones: pc:=pc+l | mar:=pc;
6.- **ldmflag;** carga de macroflags (en base al resultado de la ALU).
7.- **jump (lab);** salto incondicional a la etiqueta de nombre *lab*.
8.- **decr(pc), ldmar:** mar:=@i+1 | pc:=@destino de salto; ver junto a la explicación 4.
9.- **jump (nsal,nmz);** nmz=not maero zero, o sea, ejecuta esta instrucción de salto (de microsalto en realidad) si z=0.
10.- **0000 4080** Y bus control = 3, constant to ALU.
11.- **0051C00** Alu function = 0, (R + S)
Alu source control = 7 (R := D, S := 0), o sea, la ALU va a sumar, ¿qué va a sumar?: un 0 y la entrada externa D. ¿quién es esta entrada extema? lo vemos en el punto siguiente.
12.- **00000** constante D=0; es la inicialización que hemos puesto a *pc*.
13.- **0002 4E100** Y bus control = 2, Mem to IR.
14.- **0003 C20** sequencer instruction = 2 (JMAP).
Punto de paro activo, bit 63. Al llegar aquí el secuenciador se parará.
Corresponde con la instucción HALT que hemos puesto en el microcódigo.
15.- **0002 413100090**
0003 C2000090 ALU = NOP, no trabaja.
16.- **00410** registro fuente A y registro destino B son el reg 1: r1 = r1.
17.- **00IA 4300** sequencer instruction = CJP (conditional jump PL).
coge la dirección de salto del pipeline (PL), o sea, los bits 15-0 que si nos fijamos son 0001, la dirección del label *lab*.
18.- **01B0C0** Alu destination = B, o sea, el resultado de la ALU (F) va a parar al registro B.

Alu functíon = 0, o sea, suma pero veamos que el carry in está activo, bit 40 del pipeline.
Alu source = 3, o sea, R := 0; S := registro B.
Con todo esto lo que va a ocurrir es que B=F:=R+S+1=0+reg B+l, o sea, incrementa el registro B (debe ser el B porque aparece como fuente y destino, el A sólo puede ser fuente).

19.- **0020 4E280** Y bus function = 5, ALU to ADR.

20.- **00BB0** flag flinction = 5, Load macroflag from ALU.

21.- **0030 4E2** sequencer instruction = CONTINUE, no es una instrucción de salto, el secuenciador continuará con la siguiente microinstrucción. Comparar con la explicación 25, 26, 27 y 14.

22.- **413080** Y bus control = 1, Mem to ALU.

23.- **00002** constante utilizada en la suma, debe admitir estos 16 bits como operando para la suma por la entrada externa D (ver Alu source control).

24.- **00044** 12 bits de dirección de salto.

25.- **0041 030** /CCEN = 0, se tendrá en cuenta el bit de entrada al secuenciador /CC calculado por el FlagShifter para decidir si salta o no (apartado 2.1 del capítulo de Arquitectura). La inmensa mayoría de las microinstrucciones empiezan con 4, esto significa que /CCEN = 1 y por lo tanto se salta, no hay que preguntar a /CC; si, además, empiezan con 4E significa que se salta pero a la siguiente (sequencer instruction = E = CONTINUE).

26.- **0043 4E0** /CCEN = 1 lo que se acaba de explicar.

27.- **0044 430** /CCEN = 1, salto incondicional a la siguiente.

28.- No poner ; justo antes del *end*
Dicho *end* no lleva punto final.

29.- Diferencias generadas al utilizar los registros rO - rl5 o -registros de otros grupos (grupo 1: rl6 - r31, grupo 2: r32 - r47, grupo 3: r48 - r63). ¿cómo acceder a cualquiera de ellos? Diferencias entre ADD (que usa del grupo 0) y MOV (grupos 1 y 2).

	bits 27-22 (reg B)	bits 21-16 (reg A)
micropalabra 22: 041 =	0000 01	00 0001
micropalabra 32: 410 =	0100 00	01 0000
micropalabra 36: 820 =	1000 00	10 0000

En la micropalabra 22 se hace referencia al reg1 tanto fuente como destino; en la micro 32 es el regl6 y en la 36 es el reg32. En los tres casos los bits 29-28 del pipeline (Register Address Select) es 0 lo que indica que se toman los bits anteriores del pipeline para seleccionar los registros operandos (figura 3.2, Arquitectura de COPRO).

¿Cómo seleccionar un registro indicado por IR? Esta posibilidad será útil si pudiéramos añadir los campos dirección de operando en el IR. Supongamos una nueva instrucción:

ADD reg5, reg7; cuyo efecto es: reg5 := reg5 + reg7;

8 bits son el código de instrucción, 4 son para el registro fuente (reg7 = reg A) y cuatro para el fuente/destino (reg5 = reg B). Su microcódigo equivalente sería:

irb0 = irb0 + ira0 bits 29-16: 3000

el 3 (register address select) significa que es IR quien selecciona registros; de los ceros podemos resaltar los bits 27-26 y 21-20 para elegir el grupo 0 de registros. Estas dos últimas informaciones son los campos A y B de la figura 3.2 de Arquitectura de COPRO.

(NOTA: irb0 significa registro nº indicado por los bits 7-4 del IR dentro del grupo 0, o sea, reg0 a reg 15; lo mismo con ira0 pero con los bits 3-0 del IR).

¿Cómo usar registros superiores al reg 15? Supongamos que queremos hacer

ADD reg15, reg27 (en hexa);

en IR sólo hay 4 bits por operando, expresaremos pues ADD reg5, reg7, como antes irremediablemente. ¿Dónde establecer la diferencia entre ambos casos? en los bits 27-26 y 21-20. El microcódigo será irb1=irbl+ira2; así se tiene en cuenta el quinto registro del grupo 1 (reg 15) y el séptimo registro del grupo 2 (reg27).

5.- programa ejemplo para la macromemoria.

```
@ código          pseadocódigo

0000 0100         ADD (23),(21)
0001 0023
0002 0021
0003 V100         ADD (21),(22)
0004 0021
0005 0022
0006 0200         CUT (20),(21)
0007 0020                              código de instrucción
0008 0021
0009 040d         BEQ 10                    ADD 01
000A 0010                                   CMP 02
000B 0200         CNT (21),(21)             MOV 03
000C 0021                                   BEQ 04
000D 0021
000E 0400         BEQ 0
000F 0000
0010 0300         MOV (22),(24)
0011 0022
0012 0024
0013 0200         CMP (21),(21)
0014 0021
0015 0021
0016 0400         BEQ 13
0017 0013
                  zona de datos

0020 0005         max:=5
0021 0000         a
0022 0000         b
0023 0001         incr:=1
0024 FFFP         result
```

Este programa podría corresponder al siguiente código en Pascal:

```
a := 0; b:= 0; incr:=l; max:=5;

repeat
        a:=a+incr;
        b:=b+a;
until a=max;
result:=b;

{bucle infinito de espera}
```

7.- maqsen.mic

```
0000 4E380051C0000000
0001 4E2800B100000000
0002 4E10009040000000
0003 C200009040000000
0010 4E2801B0C0000000
0011 4E080051C0400000
0012 4E2800B100410000
0013 4E080051C0400000
0014 4E2801B0C0000000
0015 4E080051C0800000
0016 4E2800B100820000
0017 4E080051C0800000
0018 4E200A9040810000
0019 4E0001B0C0000000
001A 4300009040000001
0020 4E2801B0C0000000
0021 4E080051C0400000
0022 4E2800B100410000
0023 4E080051C0400000
0024 4E2801B0C0000000
0025 4E080051C0800000
0026 4E2800B100820000
0027 4E080051C0800000
0028 4E000BB440810000
0029 4E0001B0C0000000
002A 4300009040000001
0030 4E2801B0C0000000
0031 4E080051C4000000
0032 4E2800B104100000
0033 4E080051C4000000
0034 4E2801B0C0000000
0035 4E080051C8000000
0036 4E2800B108200000
0037 4E200A9100100000
0038 4E0001B0C0000000
0039 4300009040000001
0040 4E3800B140000002
0041 0301409040000044
0042 4E2800B2C0000000
0043 4E080051C0000000
0044 4300009040000001
```