Microensamblador de Copro II

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

Microensamblador de COPRO II

1.- Introducción

El microensamblador es un lenguaje que permite programar de una manera simple y eficaz la taijeta del procesador microprogramable Copro II. El microprograma es almacenado en una memoria llamada también micromemoria donde cada palabra direccionable de 64 bits contiene una microinstrucción ejecutable por el procesador en un periodo de reloj. Una microinstrucción es descomponible en varias microoperaciones que pueden ser ejecutadas en paralelo.

El lenguaje microensamblador de Copro II contiene instrucciones que son la representación simbólica de las microoperaciones ejecutables por el procesador. El microensamblador se encarga de la traducción en representación hexadecimal de 64 bits de las instrucciones del lenguaje microensamblador. La compilación produce un fichero objeto, que el monitor transfiere sobre la micromemoría de la tarjeta Copro II. El microensamblador evita también al programador la carga de tener que microprogrrarnar el procesador directamente, es decir escribir las microinstrucciones en binario.

2.- Especificaciones del mícroprocesador COPRO II

El microensamblador posee dos tipos de instrucciones:

* Las **pseudoinstrucciones** que son directivas dispuestas en el microcompilador para permitirle efectuar correctamente su trabajo.
* Las **instrucciones** normales que son microinstrucciones destinadas a ser compiladas y ejecutadas por el procesador.

2. 1.- Pseudoinstrucciones

Entre estas pseudoinstrucciones hay la que permite desplazar en la micromemoria el origen de las microinstrucciones: **ORG**.

La instrucción **EQU** permite la declaracion de identificadores para designar los registros accesibles del procesador.

Es también posible definir etiquetas simbólicas (labels) para designar las direcciones absolutas de saltos.

2.2- Instrucciones normales

Las instrucciones normales se clasifican en cinco categorías: Instrucciones de control del secuenciador, de control de los flags, operaciones de la unidad aritmética (que son a su vez de tres tipos: operaciones monádicas, operaciones diádicas, operaciones de desplazamiento), asl como las operaciones de transferencia de registros, de posiciones de memoria y de valores numéricos.

Las constantes numéricas pueden expresarse en base decimal hexadecimal u octal.

3.- Registros del Procesador Copro

El coprocesador Copro II posee 64 registros de utilización general en su ALU. Estos registros de 64 bits son designados por los identificadores reservados R0, RI, ..R63. Se pueden emplear en dos maneras tanto en registro A como en registro B:

Registro A:
Este registro es sólo accesible para lectura.
Sirve para almacenar el operando de una microoperación diádica en la ALU.

Registro B:
Este registro es accesible en lectura y escritura.
Sirve para almacenar:

- el operando de una microoperación
- el resultado de una operación
- el contenido de una palabra de la memoria RAM
- el contenido de otro registro
- directamente una constante numérica.

Además la ALU posee otro registro accesible bajo la designación RQ (registro Q).

Registro Q:

Este registro único es accesible en lectura y escritura, y tiene las mismas funcionalidades que un registro B.

Registro de instrucción

El procesador contiene un registro de instrucción (IR) de 16 bits que sirve para almacenar una instrucción que proviene de la RAM (16 bits).
Los 64 registros desde R0 a R63 de la ALU son accesibles por medio de este registro bajo las designaciones IRAn (n= 0, 1,2, 3) e IRBn (n = 0, 1, 2, 3) para designar al que sirve respectivamente de registro A y registro B.

Registros de control de las condiciones

Dos registros de condiciones (flag register) están disponibles para el programador:

* El registro microflag
* El registro macroflag

Estos dos registros controlan las condiciones siguientes:

* V (overflow condition)
* N (negative condition)
* Z (zero condition)
* C (carry condition)

Son accesibles bajo las designaciones UFLAG y MLAG. El registro microflag UFLAG será utilizado preferentemente para el test de las condiciones sobre el microprograma; mientras que el registro macrofiag MFLAG será utilizado para el test sobre el programa situado en la RAM.

Las operaciones posibles sobre estos registros son semejantes a las disponibles sobre los registros generales. Sin embargo es necesario tener en cuenta algunas restricciones que provienen del hecho de que su acceso moviliza el bus de datos.

Los test posibles son los siguientes:

- Test de cero (si 0 o no 0)
- Test si <0 o >0
- Test de overflow
- Test de carry
- Test de mayor que, mayor o igual
- Test de menor que, menor o igual

Registro de dirección (MAR)

Este registro sirve como registro de índice para el acceso a la memoria RAM. No se puede acceder a él más que en escritura vía la microoperación LDMAR

4.- Acceso a la memoria

La palabra de memoria apuntada por el registro de dirección se designa por la palabra reservada MEM. Para acceder a una palabra de dirección dada es, pues, necesario primeramente cargar el registro de dirección con la dirección de que se trata. Un acceso a memoria se hará con dos instrucciones. Por ejemplo: las instrucciones:

```
R0 = R0, LDMAR;
R1 = MEM;
```

permiten cargar R1 con el contenido de la palabra de memoria apuntada por R0.

5.- Juego de instrucciones del microensamblador

5.1.- Instruccíones de control

NOP

Utilización: **NOP;**
No efectúa ninguna operación, pasa a la instrucción siguiente.

HALT

Utilización **HALT;**
Detiene la ejecución de las microinstrucciones. Permite crear puntos de parada para depuración de los programas.

5.2 Instrucciones secuenciales

JMPZ

Utilización **JMPZ;**
Salta a la dirección cero de la memoria.

JMAP

Utilización: **JMAP;**
Salta en la micromemoria a la dirección definida por el registro de instrucción.

JUMP

Utilización: **JUMP (dirección);**
Salto incondicional a la dirección pasada en el parámetro.
Utilización: **JUMP (dirección, condición);**
Salto condicional a la dirección pasada en el parámetro.
Parámetros:
- Dirección: Puede ser una dirección absoluta o una etiqueta.
- Condición: UZ, NUZ, UOVR, NUOVR, UNEG, NUNEG, UC, NUC, USiG, NUSiG, USiS, NUSiS, UUnG, NUUnG, UUnS, NUUnS, MZ, NMZ, MOVR, NMOVR, MNEG, NMNEG, MC, NMC, MSiG, NMSiG, MSiS, NMSiS, MUnG, NMUnG, MUnS, NMUnS.

NOTA: La nemotecnia de las condiciones se ha construído segun los convenios siguientes:

U = micro-flag;	M = Macro-fiag;	N = Négation;
Si = Signed;	Un = Unsigned;	G = Greater, S = Smaller;
Z = Zero;	OVR = Overflow;	NEG = Negative;
C = Carry.		

CALL

Utilización: **CALL (dirección)**
Llarna al subprogrArna a la dirección expresada por el parámetro.
Utilización: **CALL (dirección, condición)**
Llama condicional al subprograma a la dirección expresada por el parárnetro.

Parámetros: Los mismos que en la instrucción JUMP

RET

Utilización: **RET**
Vuelta del subprograma
Utilización: **RET (condición)**
Vuelta condicional del subprograma

Parametros: Condicion. Los mismos códigos que en la instrucción JUMP.

LDCOUNT

Utilización: **LDCOUNT (valor)**
Carga el contador N con un valor numérico para efectuar un bucle.

Parámetro: Constante numérica.

LOOP

Utilización: **LOOP (dirección).**
Genera un bucle en la dirección expresada en el parámetro siempre que el contador N sea distinto de 0.
Parámetro: Puede ser una dirección absoluta o una etiqueta.

5.3.- Instrucciones de control de los flags

LDUFLAG

Utilización: **LDUFLAG**
Carga el registro de los microflags con los flags de la ALU.

LDMFLAG

Utilización: **LDMFLAG**
Carga el registro de los macroflags con los flags de la ALU.

5.4.- Operaciones con la ALU

5.4.1 Operaciones monádicas

INCR

Utilización: **INCR (registro).**
Incrementa en uno el contenido del registro expresado en el parámetro.
Parámetro: Registro B, RQ ó IRB.

DECR

Utilización: **DECR (registro).**
Disminuye en uno el contenido del registro expresado en el parárnetro.
Parámetro: Registro B, RQ ó IRB.

NOT

Utilización: **NOT (registro).**
Efectúa la negacion bit a bit del registro expresado en el parámetro.
Parámetro: Registro B, RQ ó IRB.

NEG

Utilización: **NEG (registro).**
Efectúa el complemento a dos del registro expresado en el parámetro.
Parámetro: Registro B, RQ ó IRB.

5.4.2 Operaciones diádicas

Destino = Operandol **operador diádico** Operando2.
Operandol, 2: Registro, palabra de memoria ó cte.
Destino: Registro, palabra de memoria.
NOTA: Todas las combinaciones entre operandos y destino no son posibles. Es necesario tener en cuenta que la utilización de un registro de flags, de una palabra de memoria o de una constante diferente de 0 monopoliza el bus de datos. Por otra parte si los dos operandos y el destino son registros generales de la ALU el registro destino debe ser el registro de uno de los operandos.

Operadores diádicos:

+

Utilización: Destino= Operando 1 + Operando 2.
Suma los dos operandos y almacena el resultado en el destino.

-

Utilización: Destino = Operandol - Operando2.
Resta los dos operandos y almacena el resultado en el destino.

AND
Utilización: Destino = Operandol **AND** Operando2.
Efectúa el AND lógico entre los dos operandos y almacena el resultado en el destino.

OR
Utilización: Destino= Operandol **OR** Operando2.
Efectúa el OR lógico entre los dos operandos y almacena el resultado en el destino.

XOR
Utilización: Destino = Operandol **XOR** Operando2.
Efectúa el XOR lógico entre los dos operandos y almacena el resultado en el destino.

XNOR
Utilización: Destino = Operandol **XNOR** Operando2.
Efectúa el XNOR lógico entre los dos operandos y almacena el resultado en el destino.

NOTRS
Utilización: Destino = Operandol **NOTRS** Operando2.
Efectúa la operacion lógica ((NOT Operandol) AND Operando2) y almacena el resultado en el destino.

COMP
Utilización: COMP (Operandol, Operando2).
Efectúa la sustracción (operandol - operando2) y carga el registro UFLAG con los flags de la ALU.
NOTA: La operación no almacena el resultado de la sustracción.

5.4.3 Operaciones de desplazamiento

Las operaciones de desplazamiento tienen lugar sobre el resultado de una operación de la ALU con cargo en el destino, así la instrucción R0 = R0, ASR; efectúa el desplazamiento aritmético a la derecha de R0.

ASR
Utilización: **ASR**
Aritmetic Shift Right (desplazamiento aritmético a la derecha).

ASL
Utilización: **ASL**
Aritmetic Shift Left (desplazamiento aritmético a la izquierda).

LS
Utilización: **LSR**
Logic Shift Right (desplazamiento lógico a la derecha).

LSL
Utilización: **LSL**
Logic Shift Left (desplazamiento lógico a la izquierda).

5.4.4 Operaciones de transferencia

Utilización: Destino - fuente
Fuente: registro, palabra de memoria ó cte.
Destino: registro, palabra de memoria ó cte.

NOTA: Todas las combinaciones entre operandos y destino no son posibles. Es necesario tener en cuenta que la utilización de un registro de flags, de una palabra de memoria o de una constante diferente de 0 monopoliza el bus de datos.

LDIR
Utilízación: **LDIR**
Carga el registro de instrucción IR con el contenido de la memoria designada por el registro de dirección.

LDMAR
Utilización: **LDMAR**

Carga el registro de dirección con el resultado de la operación de la ALU en curso. Así la instrucción INCR(R0), LDMAR; incrementa el registro R0 y carga el resultado en el registro de dirección.

5.5.- Declaración de los identificadores

EQU

Utilización: **EQU Identificador Registro;**
Efectúa la correspondencia entre el identificador y el registro a partir de su declaración en el microprograma.

NOTA: Los registros diferentes a los registros generales de la ALU no pueden su redesignados.

5.6.- Derinición de etiquetas

Se define una etiqueta haciendo preceder la microinstrucción por un identificador seguido de : . Puede por tanto servirse de esta etiqueta para designar la dirección absoluta de esta posición de memoria.

etiqueta-nombre: INCR (PZ), LDUFLAG;

5.7.- Realojamiento de las instrucciones

ORG
Utilización: **ORG dirección;**
Desplaza el origen de las microinstrucciones siguientes a partir de la dirección absoluta expresada detrás de la palabra reservada ORG.

6.- Formato de las microinstrucciones

El formato de las microinstrucciones está definido en un tamaño de 64 bis. Estos 64 bits están divididos en 16 campos controlando cada uno (más o menos) una microoperaciór. Estos campos son los siguientes: breakpoint, condition enable, data bus control, shift instruction, condition code, flag instruction, ALU carry in, ALU destination, ALU function, ALU source, register select, registerA, registreB, address, constant.

Estos dos últimos están superpuestos en el formato de la microinstrucción. Los valores de estos campos para las diversas instrucciones del lenguaje.se explican en el gráfico anexo.

7.- Estructura de un microprograma en COPRO II

Un microprograma en microensamblador COPRO II comienza por la palabra reservada **PROGRAM** y termina por la palabra reservada **END**.

Es posible poner un titulo al microprograma pero esto no es obligatorio. El titulo en este caso sigue inmediatamente a la palabra PROGRAM. Ejemplo:

PROGRAM nombre-del- programa
...
serie de instrucciones
...
END

Microinstrucción

El microprograma viene definido por una serie de microinstrucciones separadas por el caracter ";" . Varias microinstrucciones pueden ser escritas en la misma línea.

Microoperacion

Una microinstrucción puede estar compuesta por varias microoperaciones separadas cada una por el carácter ",". Estas microoperaciones deben ser independientes para que el procesador pueda ejecutarlas en paralelo. El microensamblador se encarga de verificar si no hay compatibilidad de formato en la microinstrucción. Como hay 16 campos en el formato de una microinstrucción de los cuales dos se superponen, no se pueden utilizar microoperaciones que tienen valores diferentes para el mismo campo. Es pues necesario para el programador conocer bien el formato de las microoperaciones para no cometer errores de compatibilidad de formato en cada microinstrucción.

Declaraciones

Una declaración de identificador puede hacerse en cualquier punto del programa. Además, esta declaración es válida desde este punto hasta el fin del microprograma. Una declaración de etiqueta puede encontrarse también en cualquier sitio del mieroprograma y su efecto se extiende del principio al fin del programa. Una declaración de etiqueta prevalece siempre a la de identificador.

Ejemplo:

```
 PROGRAM Ejemplo

      EQU ident_1 R0;
      EQU ident_2 Rl;
      ident 2:  INCR(ident_l);
                COMP(ident_1, ident_2), LDUFLAG;

{aquí hay un error porque ident_2 representa la etiqueta y no
el registro R2}

      JUMP (ident_1 UZ);
      HALT
 END
```

Especificaciones anexas

Los caracteres en mayúsculas y minúsculas no son distintos.

8.- Microoperaciones en paralelo

Algunas reglas simples para poder programar correctamente las microistrucciones formadas por varias microoperaciones.

Las microoperaciones se clasifican en cuatro categorias:

1.- Instrucciones de salto
2.- Instrucciones de control de flags
3.- Instrucciones de desplazamiento
4.- Operaciones con la ALU

No es posible tener más de una microoperación perteneciente a la misma categoría en una microinstrucción. Una microistrucción es pues una combinacién de microoperaciones que provienen de categorías diferentes.

Los campos de dirección y constante ocupan los mismos lugares. Ninguna núcroinstrucción pues, puede tener microoperaciones que hagan referencia a estos campos al mismo tiempo.

NOP es una operación no compatible con las demás.
HALT es una operación compatible con las demás, incluso con las microoperaciones de carga de flags LDUFLAG y LDMFLAG.

9.- Definición y sintaxis del microensamblador COPRO

Elementos para la definición del microensamblador

number ::= decimal | hexa | octal
decimal ::= decimal-digit {decimal-digit}
hexa ::= $ hexa-digit (hexa digit)
octal ::= % octal-digit (octal-digit)
decimal digit ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10
hexa-digit ::= decimal-digit | A | B | C | D | E | F
octal-digit::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0
string ::= letter {_ | letter | decimal }
letter ::= a..z | A..Z
reg ::= Rn (para n = 0..63)
IRA ::= IRA0 | IRA1 | IRA2 | IRA3
IRB ::= IRB0 | IRB1 | IRB2 | IRB3
ALUregister ::= reg | RQ | IRA | IRB
ExtRegister::= UFLAG | MFLAG
rega ::= reg | IRA
regb ::= reg | IRB
comment ::= { {character} }

Sintaxis del microensamblador

program ::= PROGRAM [title] block END
title ::= string
block ::= sentence {; sentence }
sentence ::= micro-instruction | declaration | relocation
declaration ::= EQU identifier register
relocation ::= ORG number
micro-instruction ::= [label] micro-operation {, micro-operation }
label ::= string :
micro-operation ::= NOP | HALT | jump_op | shift_op | flag_op | alu_funct_op | transfer_op
jump_op ::= JMPZ |
JMAP |
LDCOUNT (number) |
LOOP (address) |
RET [(condition)] |
CALL (address [, condition]) |
JUMP (address [, condition])
shift_op ::= ASR | ASL | LSR | LSL
flag_op ::= LDUFLAG | LDMFLAG
transfer_op ::= LDIR 1 LDMAR |
(MEM | ExtRegister) = (ALURegister | 0) |
regb = (rega | IRB | MEM | ExtRegister | Number)
alu_funct_op ::= monadic_op (regb | RQ) |
COMP (operand, operand) |
storage = operand dyadic_op operand
operand ::= storage | number
storage ::= ALUregister | ExtRegister | MEM
monadic_op ::= INCR | DECR | NOT | NEG
dyadíc_op ::= + | - | AND | OR | NOTRS | XOR | XNOR
identifier ::= string
address ::= string | number
condition ::=

UZ | NUZ | MZ | NMZ |	zero	not zero
UV | NUV | MV | NMV |	overflow	not overflow
UN | NUN | MN | NMN |	negative	not negative
UC | NUC | MC | NMC |	carry	not carry
USG | USGE | MSG | MSGE |	signed >	signed >=
USS | USSE | MSS | MSSE |	signed <	signed <=
UUG | UUGE | MUG | MUGE |	unsigned >	unsigned >=
UUS | UUSE | MUS | MUSE.	unsigned <	unsigned <=