Arquitectura y programación del MC68000

13.1. Introducción

13.2. El MC68000 visto por el programador

13.3. El programa visto por el µP MC68000

13.4.- Ejemplos

14.1.- Directivas de ensamblador y pseudoinstrucciones

14.2.- Ejemplos de realizacion de estructuras de datos

14.3.- Estructuras de programa

14.4.- Ejemplos de programación

14.5.- Conjunto instrucciones

13.1.- Introducción

MC68000

- Bus datos (16) ⇒ D0 ÷ D15
- " direcc (24) ⇒ A1 ÷ A23 ⇒ A0 - interno ⇒ apunta a los 2 bytes en memoria (8 bits)
- Registros datos (8) de 32 bits
- " direcciones (8) de 32 bits
- Operandos ALU
  - Procesadores
    - carga-almacenamiento ⇒ reg. reg.
    - Registro - memoria
    - Memoria - memoria
  - MC68000 → Híbrido
- Regular ⇒ instrucciones no presentan casos especiales
- Gran ortogonalidad ⇒ todas inst. mismos modos direcc.

13.2 El MC68000 visto por el programador

Aspectos relativos al programador

- Reg. internos
- Modos direccionamiento
- Tipos de instrucciones

- **Control**
  - **Sistema**
    - RESET
    - STOP #n — n → SR y para
    - RTE — return, habilita
    - TRAP #n — IRQ con vector n
    - MOVE Ai, USP [USP, Ai] — (Ai) → USP
    - Ins. privile → RESET, RTE, MOVE USP, MOVE to SR, ANDI, ORI, EORI sobre SR
  - **Programa**
    - Incondicional
      - BRA etiqueta — etiqueta → PC
      - JMP ax — ax → PC
      - NOP
      - BSR etiq. — (SP)-4 → SP ; (PC) → (SP) ; etiq → PC
      - JSR ax — (SP)-4 → SP ; (PC) → (SP) ; ax → PC
    - Retorno
      - RTS — ((SP)) → PC ; (SP) + 4 → SP
      - RTR — ((SP)) → CCR ; (SP) + 2 → SP ; ((SP)) → PC ; (SP) + 4 → SP
      - RTE — ((SP)) → SR ; (SP) + 2 → SP ; ((SP)) → PC ; (SP) + 4 → SP
    - Condicional
      - GT, GE, HI, CC, LT, LE, CS, LS, EQ, NE, PL
      - MI, VS, VO, T, F
      - Bcc etiq — etiq → PC
      - Scc a2 — 1....1 → a2 / 0...0 → a2
      - DBcc Di, etiq — si no cumple (Di - 1) → Di ; (Di) ≠ -1 → etiq → PC

- **Cálculo**
  - Aritméticas
  - Comparación
  - Lógicas
  - Desplazamiento
  - Modificación bits

Aritméticas

- ADD.S a_1, D_i $(a_1)+(D_i) \rightarrow D_i$
- ADDA.S a_1, A_i $(a_1)+(A_i) \rightarrow A_i$
- ADDI y ADDQ → ADDI.S #n, a_2 $n+(a_2) \rightarrow a_2$
  ADDQ.S #n, a_8 $n+(a_8) \rightarrow a_8$
- ADDX.S D_i, D_j $(D_i)+(D_j)+(X) \rightarrow D_j$
- SUB / SUBA / SUBI / SUBQ / SUBX
- NEG.S a_2 $0-(a_2) \rightarrow a_2$ / NEGX.S
- EXT.S D_i extensión bit signo
- CLR.S a_2
- MULU.W a_4, D_i sin signo / MULS.W a_4, D_i
- DIVU.W a_4, D_i / DIVS.W a_4, D_i
- ABCD D_i, D_j $(D_i)+(D_j)+(X) \rightarrow D_j$ (BCD)
- NBCD a_2 cambio signo
- SBCD D_i, D_j

Comparación

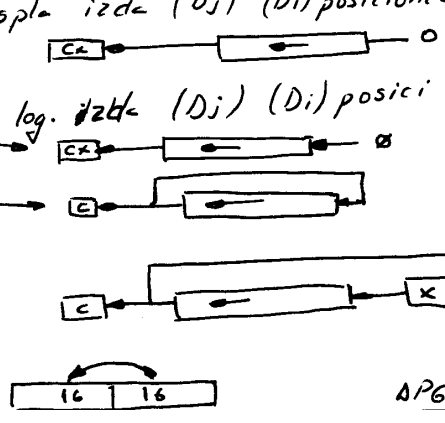
- CMP.S a_4, D_i
- CMPA a_1, A_i
- CMPI #n, a_9
- CMPM.S $(A_i)+, (A_i)+$
- TST.S a_9
- TAS.

Lógicas

- OR.S a_4, D_i $(a_4) \vee (D_i) \rightarrow D_i$
- ORI.S #n, a_2 $n \vee (a_2) \rightarrow a_2$
- EOR.S D_i, a_2 / EORI.S #n, a_2
- AND.S a_4, D_i / ANDI.S #n, a_2
- NOT.S a_2

Desplazami. rotación

- ASL.S D_i, D_j desplaz izda (D_j) (D_i) posiciones
- ASR. . . .
- LSL.S D_i, D_j " log. izda (D_j) (D_i) posici
- LSR.S D_i, D_j
- ROL.S D_i, D_j
- ROR.S D_i, D_j
- ROXL.S D_i, D_j
- ROXR.S D_i, D_j
- SWAP D_i



#n ↓

Manipulación bits

- BTST Di, a2 — bit Di a (a2) → Z
- BCLR Di, a2 — " Di de (a2) = Ø
- BSET Di, a2 — " " " " " 1
- BCHG Di, a2 — " " " " " = inverso

13.2c Modos direccionami

- Implícito → en la instrucción ⇒ RTS
- Inmediato
  - misma instrucción
  - instrucc + nº
  - MOVE.L # 11.ØØ — 11 → DØ
    - & <blanco> ⇒ decimal
    - $ ⇒ Hexadeci
    - @ ⇒ octal
    - % ⇒ bin
- Directo
  - Absoluto — MOVE.L $ØØØØ4ØFF, DØ
  - En reg. datos — ADD.L DØ, D1
  - " " direcc. — ADDA.L DØ, A1
- Indirectos
  - A registro — ADD.L (AØ), DØ
  - " " con pre-post increm — ADD.L -(AØ), DØ ; ADD.L (AØ)+, DØ
  - " " con desplaz — ADD.B $ØØ4F(AØ), DØ ; ADD.B ($ØØ4F, AØ), DØ
  - Con indice y desplaz
- Relativos a PC
  - Con desplaz.
  - con índice y desplaz

13.3. El programa visto por el procesador MC68000

Toda instrucción { C. Operación + Operando } ⇒ 2 ÷ 10 bytes { NOP ; ADD.L # $10000000, $00FFFFFF }

Ubicación en memoria

- little endian → byte menor peso en dir. mem. menor — 80x86 Intel
- big endian → " mayor peso en dir. mem. menor — MC68000

Ejecución de instrucciones

SIEMPRE PC apuntando a una instrucción ⇒ recoge C.O. lleva a decod. instrucciones, decodifica y ejecuta.

13:4 Ejemplos

Al inicio

D0 = $ 00.00.43.21
D1 = $ 12.34.56.78
D2 = $ 87.65.43.21
D3 = $ 00.FF.00.FF
D4 = $ 00.00.00.00
D5 = $ 11.11.11.11
D6 = $ 10.10.10.10
D7 = $ 01.01.01.01

A0 = $ 00.00.40.00
A1 = $ 00.00.40.0F
A7 = $ 00.00.4F.00

Memoria

$CERO	$ 4000	$00
$ UNO	$ 4001	$01
DOS	$ 4002	$02
TRES	$ 4003	$03
CUATRO	$ 4004	$00
	$ 4005	$04
CINCO	$ 4006	$0[illegible]
	$ 4007	$ 05
SEIS	$ 4008	$ 00
	$ 4009	$ 06
SIETE	$ 400A	$ 00
	$ 400B	$ 07
OCHO	$ 400C	$ 00
	$ 400D	$ 00
	$ 400E	$ 00
	$ 400F	$ 08

1.-

D0 = 00.00.43.21

MOVE.B UNO, D0

D0 = 00.00.43.00

2.-

D0 = 00.00.43.21

MOVE.W DOS, D0

D0 = 00.00.02.03

3.- D0 = 00.00.43.21

MOVE.L DOS, D0

D0 = 02.03.00.04

4.- D0 = 00.00.43.21

MOVEL UNO, D0

excepción pq UNO=impar

5.- D1 = 12.34.56.78

MOVE.L CERO+2, D1

D1 = 02.03.00.04

6.- D0 = 00.00.43.21

D1 = 12.34.56.78

ADD.B D0, D1

D0 = 00.00.43.21

D1 = 12.34.56.99

7.- D1 = 12.34.56.78

ADD.W D1, D1

D1 = 12.34.AC.F0

8.- D1 = 12.34.56.78

ADD.B D1, D1

D1 = 12.34.56.F0

9.- D1 = 12.34.56.78

ADD.L D1, D1

D1 = 24.68.AC.F0

10.- D0 = 00.00.43.21

D1 = 12.34.56.78

AND.L D0, D1

D0 = 00.00.43.21

D1 = 00.00.42.20

11.- D0 = 00.00.43.21

D1 = 12.34.56.78

AND.W D0, D1

D1 = 12.34.42.20

12.- D0 = 00.00.43.21

D1 = 12.34.56.78

OR.L D0, D1

D0 = 12.34.57.79

13.- D6 = 10.10.10.10

D2 = 87.65.43.21

OR.L D6, D2

D6 = 10.10.10.10

D2 = 97.75.53.31

14.- D1 = 12.34.56.78

EOR.W D1, D1

D1 = 12.34.00.00

13.5 La pila del MC68000

Conjunto de posiciones de memoria de tamaño variable, utilizable para almacenar información antes de los saltos a las subrutinas.

Elementos:
- Cabeza de la pila → Reg. Puntero de Pila (SP) (A7)
- Memoria

clases:
- sistema SSP
- usuario USP

Instrucciones:
- Meter → MOVE.x Dn, -(A7)
- Sacar → MOVE.x (A7)+, Dn

13.6 Interrupciones y excepciones

Excepción = suspensión de la ejecución de la secuencia de instrucciones en curso.

Tipos:
- Interno
  - errores direccionamiento
  - inst. desconocidas
  - " no permitidas
- Externo
  - RESET
  - error de BUS
  - interrupciones (periféricos)

Concepto: La CPU está ejecutando la secuencia normal de ejecución del programa. Aparece una excepción (se produce una señal hardware) y la CPU reacciona de la siguiente manera:

1º Termina la instrucción en curso

2º Realiza una serie de acciones preprogramadas internamente. (Guardar parámetros en pila y otros) El valor del PC se mete en la pila

3º Carga en el PC la dirección asociada a la excepción producida ("Vector interrupción")

4º En dicha subrutina se guardan en la Pila los registros que se modifican en ella y que luego ha de disponerse de ellos.

5º Se termina la ejecución de la subrutina, devolviendo desde la pila los valores guardados al principio

6º Se recupera el valor del PC previamente almacenado en la pila (RTE), para continuar con el programa interrumpido

Prioridades y habilitaciones: Existe una estructura de prioridades y habilitaciones que permite atender antes a una excepciones (interrupciones) que a otras.

14.- Ensamblador para el MC68000

Lenguaje ensamblador = lenguaje compuesto por nemotécnicos y muy ligado a la constitución del sistema.

Sintaxis

[<etiqueta>] [<mnemo_instru> [<operandos>]] ; [<comentarios>]

Directivas

- ORG <dir. o expres> → Origen absoluto de las instruc. o datos que le sigon
- END → Fin prog. fuente
- EQU <etiq> EQU <valor o expres> → asigna el valor <val> a la <etiq>
- DS <etiq> DS.x <nº varia> → reserva <nº varia> posic. de mem.
- DC <etiq> DC.x <val o valores> → reserva mem y mete los valores

```
Longitud    EQU   4        ; longitud tiene valor 4
valor1      EQU   25       ; valor1 = 25 = 19 hex
valor2      EQU   $A9

; Aquí empieza el programa principal
            ORG   $400     ; empieza en la dir 400 hexade
            NOP            ; inst. NOP se graba en 400 hex
; Zona subrutinas
            ORG   $C00
            NOP            ; NOP se graba en C00 hex
```

```
; zona de datos
        ORG     $1000
vector  DS.W    longitud    ; se reservan 4 pos. mem a partir
                            ; de la 1000 hex
cont1   DC.B    valor1,valor2 ; se guarda en la pos 1004 hex
                            ; el dato 19 hex y en la 1005 el A9 hex
cont2   DC.L    valor1, valor2 ; igual a la anterior
        END                 ; fin programa
```

14.3 Estructuras de programas

1.- Secuencias → sucesión de instrucciones

2.- Bifurcaciones

```
IF <condicion> THEN [secuencia 1];
               ELSE [secuencia 2];
```

En ensamblador

condición ⇒ comparación entre var1 y var2

```
MOVE.L  var1, D0;        MOVE.L  var1,D0
CMP.L   var2, D0;        CMP.L   var2,D0
Bcc     sec1             Bcc     sec2
BRA     sec2             [secuencia 1]
sec1: [secuencia 1]      sec2: [secuencia 2]
sec2: [secuencia 2]
```

3.- Iteraciones

FOR

FOR (contador=1) TO n DO [secuencia 1]

[secuencia 2]

```
contador    EQU    #12
            MOVE.L  #contador,D4   ; carga 12 = nº iter.
            MOVE.L  #0,D5
bucle:      ADQ.L   #1,D5          ; increm. D5
            [secuencia 1]
            CMP.L   D5,D4          ; compara si 12 increm
            BLT     bucle          ; salta si D5<D4 a
                                   ; bucle
            [secuencia 2]
```

→

```
            MOVE.L  #contador,D5
bucle:      [secuencia 1]
            DBF     D5,bucle       ; decre D5 y si falso=0
            [secuencia 2]          ; salta a bucle
```

WHILE

```
WHILE (condición) DO
        [secuencia 1]
END of WHILE
[secuencia 2]
```

condición → var1 y var2

```
        MOVE.L  var1, D0
        MOVE.L  var2, D1
bucle : CMP.L   D0, D1
        Bcc     fin_bucle
        [secuencia 1]
fin_bucle : [secuencia 2]
```

REPEAT - UNTIL

```
REPEAT
        [secuencia 1]
UNTIL (var1 ≠ var2)
[secuencia 2]
```

```
        MOVE.L  var1, D0
        MOVE.L  var2, D1
bucle : [secuencia 1]
        CMP.L   D0, D1
        Bcc     bucle
        [secuencia 2]
```

LOOP

```
        MOVE.L   var1,D0
        MOVE.L   var2,D1
bucle:  [secuencia 1a];
        CMP.L    D0,D1
        Bcc      fin_bucle
        [secuencia 1b]
        CMP.L    D0,D1
        Bcc      fin_bucle
        [secuencia 1c]
        ;
        BRA      bucle
fin_bucle: [secuencia 2]
```