Departament de Disseny i Programació de Sistemes Electrònics
UNIVERSITAT POLITÈCNICA DE CATALUNYA

Arquitectura de computadors

Pràctica 5

Disseny del camí de dades

Antoni Escobet

PRÀCTICA 5 – Disseny del camí de dades

1. Objectius

En aquesta practica es pretén:

- Repassar els conceptes relacionats amb la unitat central de processos d'un sistema microprocessador MIPS.
- Dissenyar el camí de dades (conjunt unitat central de processos (CPU) més la memòria) estudiada a la classe de teoria.
- Amplia els coneixements sobre el llenguatge VHDL i l'edició amb QUARTUS II.

2. Material

L'únic material necessari per la realització d'aquesta pràctica és el paquet de programari d'ALTERA, el Quartus II web edition instal·lat als ordinadors del laboratori, i que us podeu descarregar gratuïtament de la pàgina web d'Altera (http://www.altera.com).

Recordeu baixar-vos el programa de simulació: ModelSim-Altera Edition per la versió de Quartus II que tingueu instal·lada.

3. Problema proposat

En aquesta pràctica s'ha de completar el disseny del processador que es pot veure a la figura 1.

En les pràctiques anteriors s'han desenvolupat alguns dels elements més importants del camí de dades: l'ALU, el banc de registres, el comptador de programa i la unitat de control. En falten altres com:

- Extsig, que és l'encarregat de convertir el valor immediat de 16 bits de les instruccions de tipus I en un valor de 32 bits estenent-li el signe.
- Dos << 2, que són desplaçadors de dos bits. Un s'utilitza per generar la direcció real del salt (de 32 bits) a partir de la dada de 26 bits inclòs en les instruccions de salt incondicional (de tipus J). I l'altre, per a generar *l'òffset* que cal sumar al comptador de programa per calcular l'adreça real del salt (de 32 bits) a partir de la dada de 16 bits inclòs en les instruccions de salt condicional (de tipus I).
- El registre d'instruccions (IR), que funcionalment no és més que un biestable (latch) (D) de 32 bits. És a dir, és un "registre" habilitat per nivell, i sense control de sortida de tres estats (la sortida d'aquest registre sempre és visible).
- Un multiplexor de 4 a 1, la missió del qual és seleccionar el registre destí en funció del tipus d'instrucció.
- PCSUP, que genera una còpia del comptador de programa mantenint únicament els quatre bits de més pes, i posant a '0' els 28 bits de menys pes.
- El registre acumulador (Acc). Igual que el registre d'instruccions, és un biestable de 32 bits, però amb control de sortida amb tres estats.
- Els registres d'accés a la memòria: MAR i MDR. El MAR (o registre d'adreces) és, un biestable de 32 bits. El MDR (o registre de dades), és un disseny més complex per dos motius: El primer, perquè l'entrada d'informació cap al registre pot procedir del bus 2 (en accessos d'escriptura) o de la memòria (en accessos de lectura). I el segon, la sortida de la informació pot anar dirigida cap al bus 3 (per lectures) o cap a la memòria (per escriptures). És a dir, té una entrada (connectada al bus 2), una sortida (connectada al bus 3) i una connexió bidireccional amb la sortida de dades de la memòria.
- L'enllaç de bus que connecta els busos 1 i 3, utilitzat bàsicament per a accedir a la memòria (al registre MAR) amb l'adreça efectiva calculada a l'ALU.
- La memòria RAM. Per simplificar el disseny, i degut a que els programes que es volen fer no són excessivament llargs, s'ha implementat una memòria de només 128 bytes, o el que és el mateix, 32

paraules de 32 bits. A *priori*, la memòria ja conté un senzill programa en assemblador que haureu d'esbrinar que fa a partir de la simulació del sistema complet.

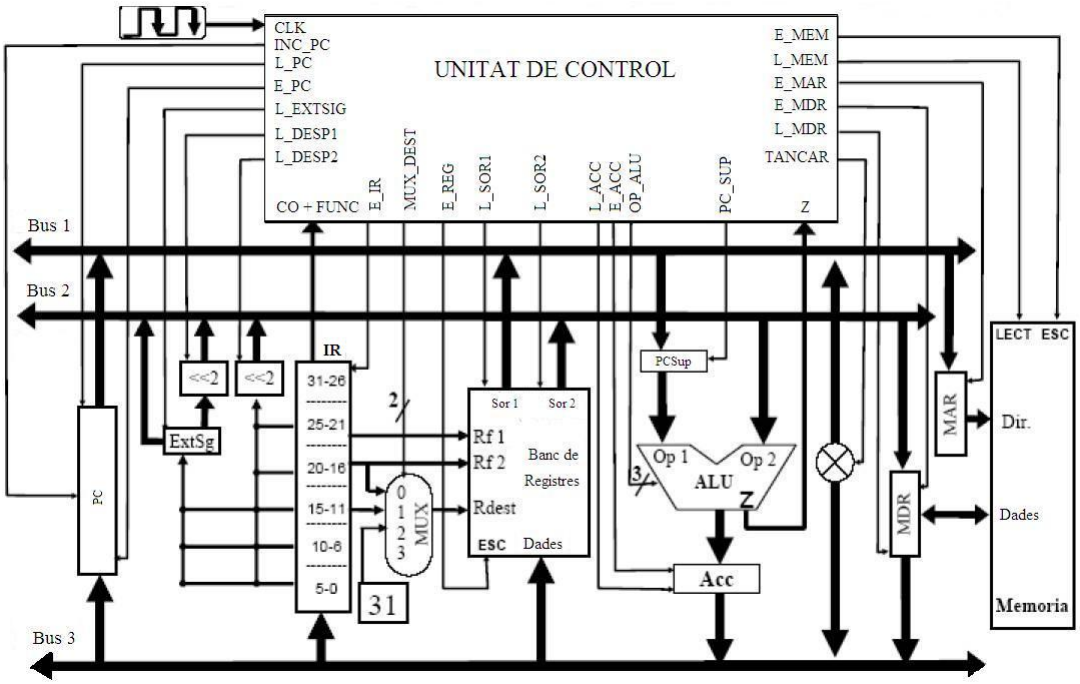

Figura 1

Els dissenys de tots aquests elements estan inclosos, junt amb l'esquelet del disseny del camí de dades, al fitxer **PRAC5.ZIP**, disponible a la web de l'assignatura (Atenea).

Convé recordar el joc d'instruccions del processador que s'està dissenyant:

Operacions aritmètiques - lògiques	
ADD	Suma aritmètica
SUB	Resta
AND	Producte lògic
OR	Suma lògica
SLT	Comparació: menor que
ADDI	Suma aritmètica immediata
ANDI	Producte lògic immediat
ORI	Suma lògica immediata
Operacions de transferència amb memòria	
LW	Càrrega paraula de memòria
SW	Emmagatzemament de paraula a memòria
Operacions de ruptura de seqüència	
BEQ	Salt condicional (si igual)
BNE	Salt condicional (si diferent)
J	Salt incondicional
JAL	Salt subrutina
JR	Salt indirecte

4. Descripció de la practica

En aquesta secció es descriu com realitzar el disseny del processador. En primer lloc es presenta la definició de l'entitat a realitzar. Tindrà la següent definició d'entrades:

```
entity Microprocessador Is
    Port (
        Reset : in STD_LOGIC;
        CLK : in STD_LOGIC );
end Microprocessador;
```

En principi, s'ha dissenyat el sistema microprocessador format per tots els elements que conformen la CPU més la memòria. Evidentment, en una visió més realista del sistema es podria haver implementat per un costat la CPU (integrant tots els components mencionats anteriorment excepte la memòria), la pròpia memòria (RAM), així com altres perifèrics que es pogueren afegir: altres memòries, controladors d'interrupcions, ports de E/S, i un llarg etcètera. Però no s'ha fet perquè l'objectiu principal buscat amb aq1uestes pràctiques és que pugueu implementar el camí de dades que s'ha vist a teoria, i no aprofundir en l'estructura del computador.

La descripció dels components que necessiteu és:

```
component ALU32 is
    Port (Op1 : in  STD_LOGIC_VECTOR (31 downto 0);
          Op2 : in  STD_LOGIC_VECTOR (31 downto 0);
          Operacio : in  STD_LOGIC_VECTOR (2 downto 0);
          Result : out  STD_LOGIC_VECTOR (31 downto 0);
          Z,C : out  STD_LOGIC );
end component;
```

```
component BancRegistres is
   Port ( Reset : in STD_LOGIC;
          clk : in STD_LOGIC;
          esc : in STD_LOGIC;
          rdest : in STD_LOGIC_VECTOR (4 downto 0);
          dades : in STD_LOGIC_VECTOR (31 downto 0);
          l_sor1 : in STD_LOGIC;
          rf1 : in STD_LOGIC_VECTOR (4 downto 0);
          l_sor2 : in STD_LOGIC;
          rf2 : in STD_LOGIC_VECTOR (4 downto 0);
          sor1 : out STD_LOGIC_VECTOR (31 downto 0);
          sor2 : out STD_LOGIC_VECTOR (31 downto 0) );
end component;
```

```
component PCSUP is
  Port ( pc_sup : in std_logic;
       Entrada : in  STD_LOGIC_VECTOR (31 downto 0);
       Sortida : out  STD_LOGIC_VECTOR (31 downto 0) );
end component;
```

```
component ComptadorDePrograma is
    Port ( Reset : in STD_LOGIC;
           clk: in STD_LOGIC;
           Inc_PC : in STD_LOGIC;
           L_PC : in STD_LOGIC;
           E_PC: in STD_LOGIC;
           Din: in STD_LOGIC_VECTOR (31 downto 0);
           Dout: out STD_LOGIC_VECTOR (31 downto 0) );
end component;
```

```
component Desp25a0 is
  Port ( l_desp2 : in std_logic;
         Entrada : in STD_LOGIC_VECTOR (25 downto 0);
         SortidaBus : out STD_LOGIC_VECTOR (31 downto 0) );
end component;
```

```
component DespExtSign15a0 is
  Port ( l_desp1 : in STD_LOGIC;
         Entrada : in STD_LOGIC_VECTOR (31 downto 0);
         SortidaBus : out STD_LOGIC_VECTOR (31 downto 0) );
end component;
```

```
component ExtensioDeSigne is
  Port ( l_extsign: in STD_LOGIC;
         DadaEntrada : in  STD_LOGIC_VECTOR (15 downto 0);
         DadaSortida1 : out  STD_LOGIC_VECTOR (31 downto 0);
         DadaSortida2 : out STD_LOGIC_VECTOR (31 downto 0) );
end component;
```

```
component RAMSin is
  Port ( clk : in STD_LOGIC;
           Escr : in STD_LOGIC;
           Lect : in STD_LOGIC;
           Adress  : in STD_LOGIC_VECTOR (5 downto 0);
           DadesE : in  STD_LOGIC_VECTOR (31 downto 0);
           DadesS : out  STD_LOGIC_VECTOR (31 downto 0) );
end component;
```

```
component Mux4a1_5Bits is
  Port ( Sel : in  STD_LOGIC_VECTOR (1 downto 0);
         E0 : in  STD_LOGIC_VECTOR (4 downto 0);
         E1 : in  STD_LOGIC_VECTOR (4 downto 0);
         E2 : in  STD_LOGIC_VECTOR (4 downto 0);
         E3 : in  STD_LOGIC_VECTOR (4 downto 0);
         S : out  STD_LOGIC_VECTOR (4 downto 0) );
end component;
```

```
component Acumulador is
  Port ( clk : in STD_LOGIC;
           l_acc : in  STD_LOGIC;
           e_acc : in STD_LOGIC;
           DadaEntrada : in  STD_LOGIC_VECTOR (31 downto 0);
           DadaSortida : out  STD_LOGIC_VECTOR (31 downto 0) );
end component;
```

```
component RegistreMDR is
  Port ( clk : in STD_LOGIC;
           reset : in STD_LOGIC;
           e_mdr : in  STD_LOGIC;
           l_mdr : in  STD_LOGIC;
           l_mem : in STD_LOGIC;
           DadesBus2 : in  STD_LOGIC_VECTOR (31 downto 0);
           DadesMemS : out  STD_LOGIC_VECTOR (31 downto 0);
           DadesMemE : in  STD_LOGIC_VECTOR (31 downto 0);
           DadesBus3 : out STD_LOGIC_VECTOR (31 downto 0) );
end component;
```

```
component RegistreMAR is
  Port (
        Pas : in  STD_LOGIC;
        DadesEnt : in STD_LOGIC_VECTOR (31 downto 0);
        DadesSor : out STD_LOGIC_VECTOR (31 downto 0) );
end component;
```

```
component Latch32Bits is
  Port ( clk: in STD_LOGIC;
        E : in  STD_LOGIC;
        reset : in STD_LOGIC;
        Din : in  STD_LOGIC_VECTOR (31 downto 0);
        Dout : out  STD_LOGIC_VECTOR (31 downto 0) );
end component;
```

```
component UnitatDeControl is
    Port ( Reset : in STD_LOGIC;
        clk : in  STD_LOGIC;
        co : in  STD_LOGIC_VECTOR (5 downto 0);
        func : in  STD_LOGIC_VECTOR (5 downto 0);
        z : in  STD_LOGIC;
        l_sor1 : out  STD_LOGIC;
        l_sor2 : out  STD_LOGIC;
        e_reg : out  STD_LOGIC;
        mux_dest : out  STD_LOGIC_VECTOR (1 downto 0);
        l_mem : out  STD_LOGIC;
        e_mem : out  STD_LOGIC;
        e_mar : out  STD_LOGIC;
        l_mdr : out  STD_LOGIC;
        e_mdr : out  STD_LOGIC;
        e_ir : out  STD_LOGIC;
        l_pc : out  STD_LOGIC;
        e_pc : out  STD_LOGIC;
        pc_sup : out  STD_LOGIC;
        inc_pc : out  STD_LOGIC;
        l_desp1 : out  STD_LOGIC;
        l_desp2 : out  STD_LOGIC;
        l_extsign : out  STD_LOGIC;
        l_acc : out  STD_LOGIC;
        e_acc : out  STD_LOGIC;
        Tancar : out  STD_LOGIC;
        op_alu : out  STD_LOGIC_VECTOR (2 downto 0)
    );
end component;
```

```
component EnllacBus is
  Port ( Tancar : in  STD_LOGIC;
        DadesBus3 : in STD_LOGIC_VECTOR (31 downto 0);
        DadesBus1 : out STD_LOGIC_VECTOR (31 downto 0) );
end component;
```

5. Realització pràctica

5.1. Exercici 1

En aquets apartat s'indiquen alguns passos previs a la pròpia realització de la pràctica:

1. Descomprimiu el fitxer **PRAC5.ZIP** sobre el directori de treball. Aquest fitxer conté els fitxers font (VHDL) de tots el models menys els que ja heu realitzat en les pràctiques anteriors.
2. Copieu sobre el directori de treball TOTS els fitxers font realitzats a les pràctiques anteriors: l'ALU, el banc de registres, el comptador de programa i la unitat de control.

3. Editeu els fitxers propis i modifiqueu-los, si es necessari, perquè siguin compatibles amb els components importats en el disseny del processador.
4. Obriu l'eina Quartus II d'Altera i crear el projecte pel processador, seguint els mateixos passos que en les pràctiques anteriors.
5. Afegiu al projecte tots els fitxers font.
6. Creeu els símbols de tots els components que necessiteu

5.2. Exercici 2

Feu el disseny de l'esquemàtic del microprocessador. Cal recordar que la grandària real de la memòria és de 128 bytes, per la qual cosa al bus d'adreces de la memòria se li connectaran únicament els 7 bits de menor pes del bus d'adreces (el Bus1).

A continuació, compileu el model i assegureu-vos de que no hi ha cap error

5.3. Exercici 3

Simuleu el projecte. Per fer-ho, s'ha de crear un fitxer de simulació (Test Bench Waveform) amb les opcions següents:

```
clock : PROCESS
BEGIN
        clk <= '0';
        WAIT FOR 50 ns;
        CLK <= '1';
        WAIT FOR 50 ns;
end process clock;
```

```
always : PROCESS
BEGIN
        RESET <= '1';
        WAIT FOR 120 ns;
        RESET <= '0';
        WAIT FOR 20 us;
WAIT;
END PROCESS always;
```

En la traça apareixeran únicament els ports Reset i CLK, que son els únics ports del microprocessador. Per a poder realitzar una simulació completa (mostrant un conjunt més ampli de senyals, com per exemple l'estat de la unitat de control, els busos, el comptador de programa, el registre d'instruccions o els registres del banc) caldrà realitzar, en primer lloc, la simulació i a continuació, i després d'executar la simulació Simulate Behavioral Model, sobre la traça es poden afegir tots els senyals que es consideri oportú.

A partir de la simulació, heu de deduir quin programa s'ha executat, i ompli el següent codi en assemblador (sense utilitzar directives):

```
.data 0x
.word

. text 0x00000000
```

Ompliu la següent taula indicant quin és el contingut (en hexadecimal) dels busos i dels registres utilitzats pel programa en l'última fase de l'última instrucció (al voltant dels 20 us).

	Valor (hex)
Bus1	0x
Bus2	0x
Bus3	0x
Registre	0x
Registre	0x
Registre	0x
Registre	0x

En tots els casos, heu de presentar el codi VHDL realitzat per a cadascun dels exercicis, així com les simulacions que demostren el funcionament correcte dels circuits realitzats