Departament d'Enginyeria Minera, Industrial i TIC

Arquitectura de computadors

Pràctica 3

22 de març i 5 d'abril de 2017

Disseny de la CPU

Antoni Escobet

PRÀCTICA 3 – Disseny de la CPU

1. Objectius

En aquesta practica es pretén:

- Repassar els conceptes relacionats amb la unitat de control d'un processador MIPS monocicle i de la resta d'unitats funcionals de la CPU.
- Dissenyar la unitat de control.
- Dissenyar la unitat de control de l'ALU.
- Interpretar la resta d'unitats funcionals de la CPU
- Dissenyar el camí de dades de la CPU i la memòria.
- Amplia els coneixements sobre el llenguatge VHDL i l'edició amb QUARTUS.

2. Material

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

La simulació del vostre disseny s'ha de fer amb el simulador que proporciona el mateix paquet de programari d'Altera (ModelSim-Altera)

3. Problema proposat

Aquesta practica està formada per dues parts clarament diferenciades. A la primera part, s'ha de realitzar el disseny de la unitat de control pel processador monocicle vist a classe. Tal com s'ha vist, la UC és la part del processador que dirigeix i coordina totes les operacions que realitza la CPU. A més a més, en aquesta part també s'hi afegeix la unitat de control de l'ALU, que tal i com es va explicar, podria formar part de la unitat de control, però ho farem en un bloc a part.

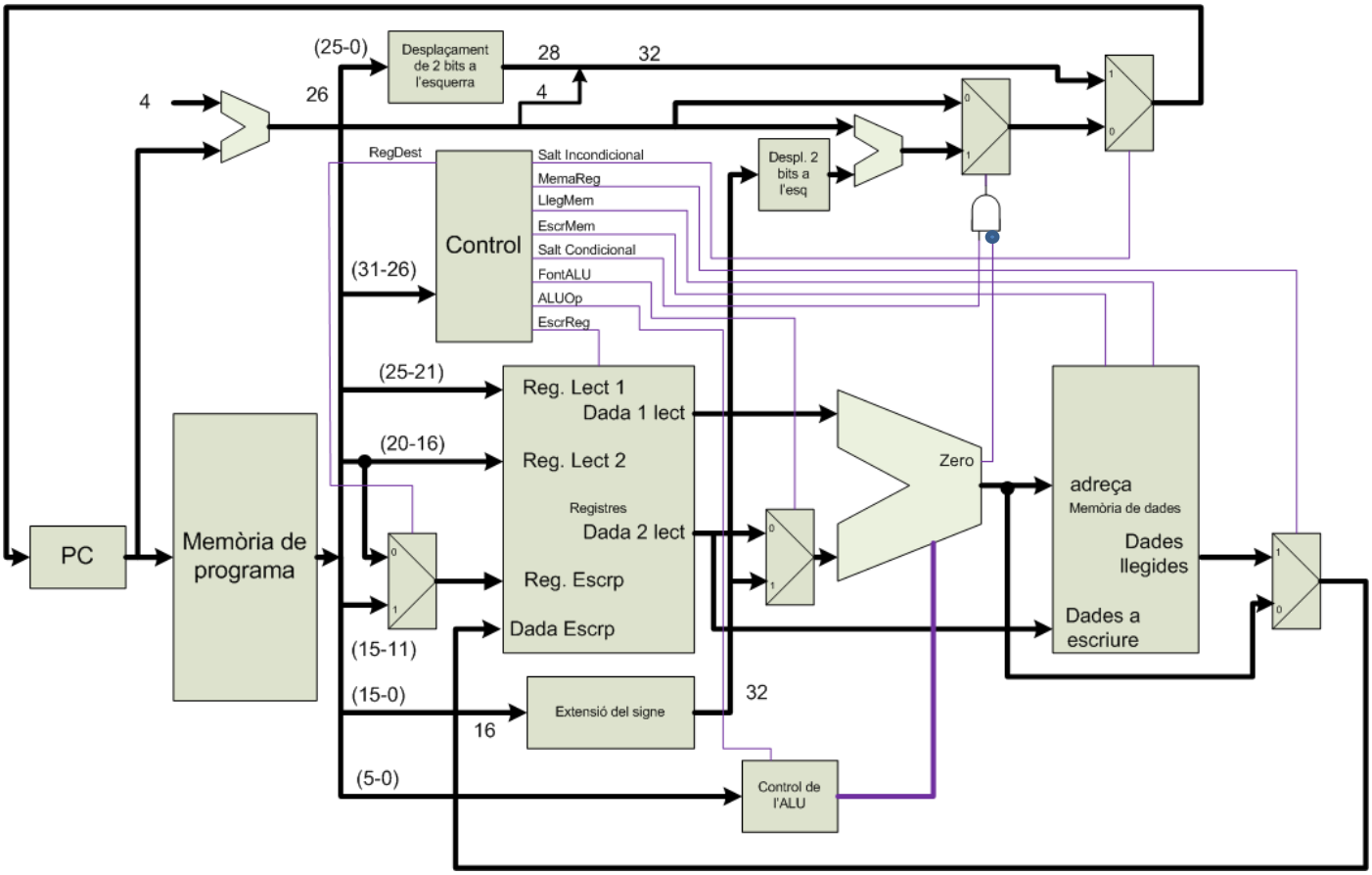
La segona part de la practica, consisteix es el disseny complert del camí de dades del processador. Per tal de poder fer aquest disseny, teniu penjat la resta de components que necessiteu. L'únic que heu de fer amb aquest components, és interpretar-los i connectar-los amb els components que heu anat dissenyant en les practiques anteriors.

El disseny del processador vist a classe, amb el seu camí de dades i la seva unitat de control permet executar un conjunt reduït d'instruccions relacionades amb la unitat aritmètica i lògica, es a dir, que pot realitzar sumes i restes de nombres sencers de 32 bits, i les operacions lògiques AND i OR sobre 32 bits. També pot comparar dos operants per saber si un és major que l'altre.

Per tal de poder fer els salts condicionals, serà necessari que l'ALU proporcioni a la unitat de control, l'indicador de si el resultat de l'operació val zero (Z).

Les instruccions aritmètiques i lògiques necessiten utilitzar operands ubicats al banc de registres. Inicialment les instruccions i les dades del programa es troben a la memòria. Per tant, abans de poder utilitzar un dada del programa serà necessari portar-ho des de memòria a un registre determinat. A més, és possible que necessitem escriure a la memòria algun resultat emmagatzemat en un registre. Per portar una dada des de la memòria a un registre es necessita una operació de lectura sobre memòria. De la mateixa forma, per emmagatzemar un dada continguda en un registre a la memòria s'ha de fer una operació d'escriptura sobre memòria. L'arquitectura MIPS ofereix dues instruccions específiques per a realitzar la lectura d'una paraula de memòria

i l'escriptura o emmagatzemament d'una paraula en memòria. Aquestes són: lw, *load word i* sw, *store word*, respectivament.



El joc d'instruccions del processador a dissenyar, és:

Operacions aritmètiques - lògiques	
ADD	Suma aritmètica
SUB	Resta
AND	Producte lògic
OR	Suma lògica
SLTI	Comparació: menor que
ADDI	Suma aritmètica immediata
Operacions de transferència amb memòria	
LW	Càrrega paraula de memòria
SW	Emmagatzemament de paraula a memòria
Operacions de ruptura de seqüència	
BNE	Salt condicional (si no és igual)
J	Salt incondicional
Varis	
NOP	Operació que no fa res

4. Descripció de la practica

En aquesta secció es descriu com realitzar les diferents parts de la practica. En primer lloc, després de recordar l'estructura estudiada a classe sobre la relació entre les diverses unitats funcionals del processador, presentarem la definició de les entitats VHDL a realitzar.

4.1. Unitat de Control

La UC té les següents entrades:

- El camp “codi d'operació” CO del bus d'instruccions (IR) (bits 31..26)
- Els indicadors ‘Z’ i ‘C’ de l’ALU (A l’esquema només hi ha la Z)

Recordeu que el format de les instruccions del processador és:

	31 26	25 21	20 16	15 11	10 6	5 0
Tipus R	CO	rs	rt	rd	0	func
	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

	31 26	25 21	20 16	15 0
Tipus I	CO	rs	rt	Valor
	6 bits	5 bits	5 bits	16 bits

	31 26	25 0
Tipus J	CO	adreça
	6 bits	26 bits

Els codis d'operació (CO) de les diferents instruccions que executa el processador, son:

	CO								
Instrucció	**C5**	**C4**	**C3**	**C2**	**C1**	**C0**		Tipus	ALUOp
add, sub, and, or, nop	0	0	0	0	0	0	0x00	R	10
addi	0	0	1	0	0	0	0x08	I	00
lw	1	0	0	0	1	1	0x23	I	00
sw	1	0	1	0	1	1	0x2B	I	00
bne	0	0	0	1	0	0	0x04	I	01
slti	0	0	1	0	1	0	0x0A	I	11
J	0	0	0	0	1	0	0x02	J	xx

La missió de la unitat de control, és generar els senyals que controlen els elements dels processador. Aquest senyals estan descrit a les transparències de classe.

```
entity UnitatControl is
Port (
    co : in STD_LOGIC_VECTOR (5 downto 0);
    z : in STD_LOGIC;
    c : in STD_LOGIC;
    RegDest : out STD_LOGIC;
    SaltInc : out STD_LOGIC;
    MemaReg : out STD_LOGIC;
    LlegMem : out STD_LOGIC_VECTOR (1 downto 0);
    EscrMem : out STD_LOGIC;
    SaltCon : out STD_LOGIC;
    FontALU : out STD_LOGIC;
    ALUOP : out STD_LOGIC_VECTOR(1 downto 0) ;
    EscrReg : out STD_LOGIC;
);
end UnitatControl;
```

La unitat de control, és un sistema combinacional simple, que ha de generar les sortides, sempre que hi hagi un canvi a les entrades.

4.2. Control ALU

Aquesta unitat de control de l'ALU, ha de generar els tres bits de comandament que té la nostre ALU. Les entrades d'aquesta unitat, seran els dos bits generats per la UC dissenyada a l'apartat anterior, i els 6 bits del camp de "funció" de la instrucció.

Els codis d'operació (CO) i funció (func) són els següents per a les diferents instruccions que executa el processador:

	c.o.						**Funció**					
Instrucció	**C5**	**C4**	**C3**	**C2**	**C1**	**C0**	**F5**	**F4**	**F3**	**F2**	**F1**	**F0**
add	0	0	0	0	0	0	1	0	0	0	0	0
sub	0	0	0	0	0	0	1	0	0	0	1	0
and	0	0	0	0	0	0	1	0	0	1	0	0
or	0	0	0	0	0	0	1	0	0	1	0	1
nop	0	0	0	0	0	0	0	0	0	0	0	0
slti	0	0	1	0	1	0	x	x	x	x	x	x
addi	0	0	1	0	0	0	x	x	x	x	x	x
lw	1	0	0	0	1	1	x	x	x	x	x	x
sw	1	0	1	0	1	1	x	x	x	x	x	x
bne	0	0	0	1	0	0	x	x	x	x	x	x
J	0	0	0	0	1	0	x	x	x	x	x	x

Una part del seu disseny el teniu descrit a les transparències de classe a la descripció del processador monocicle.

```
entity ControlALU is
Port (    funcio : in STD_LOGIC_VECTOR (5 downto 0);
          CodiOP : in STD_LOGIC_VECTOR (1 downto 0);
          ControlALU : out STD_LOGIC_VECTOR (2 downto 0);
); end ControlALU;
```

4.3. Disseny de la CPU

En els apartats i pràctiques anteriors heu desenvolupat alguns dels elements més importants del camí de dades: ALU, el Banc de registres, el comptador de programa, la unitat de control i la unitat de control de l'ALU. En falten altres com:

- ExtSig.vht: 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.
- Sumador.vhd: Genera la suma de dues entrades de 32 bits. Se'n necessiten dos, un per sumar el PC + 4 i l'altre per poder executar les instruccions de salt.
- Mux2x1.vht: Multiplexor de dos busos de 32 bits. Es necessita en diferents llocs del camí de dades.
- Mux2x1_5bits.vht : Multiplexor de dos bussos de 5 bits. Es necessita per poder seleccionar el registre de destí.
- MemData.vht : Memòria de dades.
- MemInst.vht : Memòria de programa.

La definició de l'entitat de la CPU, pot ser:

```
entity CPUMonocicle 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 les 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 les memòries), la memòria de rograma i, la memòria de dades (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 aquestes pràctiques és que pugueu implementar el camí de dades que s'ha vist a teoria, i no aprofundir en l'estructura del computador.

5. Realització pràctica

5.1. Exercici 1

Dissenyeu la unitat de control de la CPU. A continuació s'ha de comprovar mitjançant la simulació que el disseny funciona correctament. Per fer-ho es realitzaran un conjunt de simulacions comportamentals mitjançant el simulador que trieu. Cadascun representarà la execució separada de les instruccions **add**, **lw** i **bne**.

5.2. Exercici 2

Dissenyeu la unitat de control de l'ALU, i demostreu que funciona correctament amb el simulador. Feu més d'una prova, donant diferents valors (que siguin realistes) a les entrades. Comproveu que es generen les sortides que pertoquen en cadascun dels casos.

5.3. Exercici 3

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

1. Descomprimiu el fitxer **PRAC3.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 les unitats 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.

5.4. Exercici 4

Feu el disseny del microprocessador, compileu el model i assegureu-vos de que no hi ha cap error.

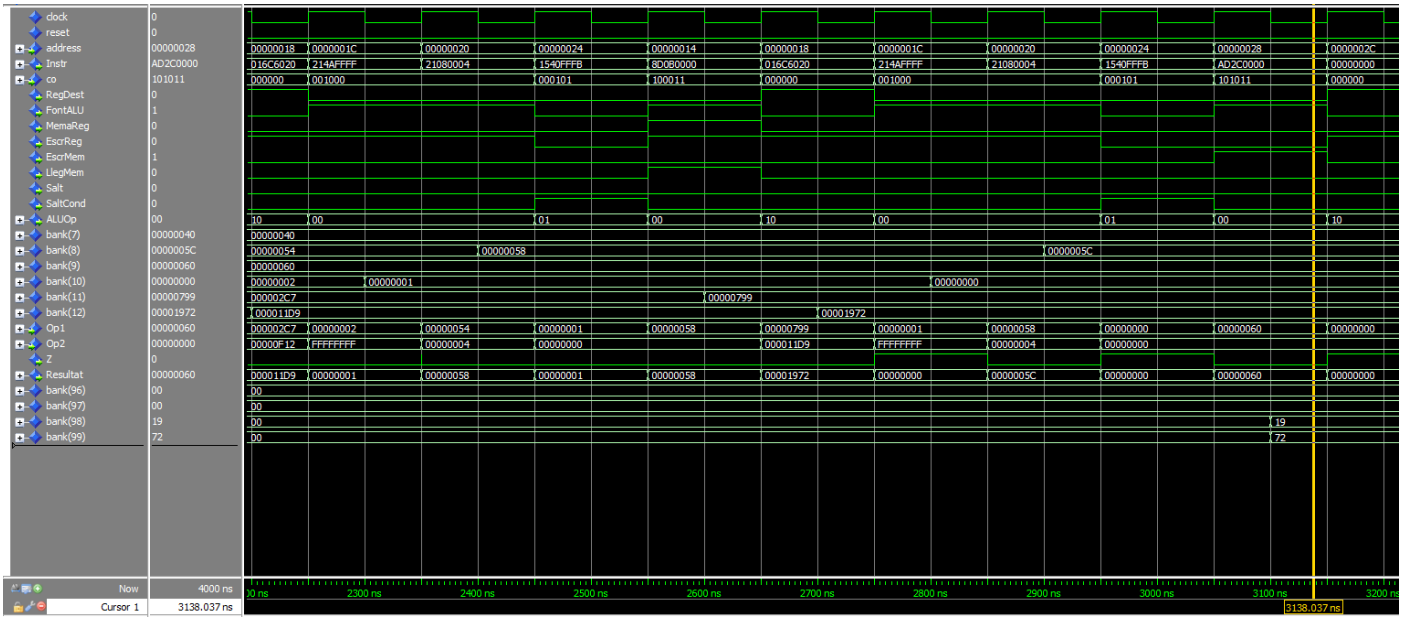
5.5. Exercici 5

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 3200 ns;
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 heu d'afegir, com a mínim els senyals : Valor del PC, Instrucció que s'està executant, sortida de l'ALU, adreça de la memòria de dades, els registres del 7 al 13, i les posicions de memòria de dades de la 0x40 a la 0x60, a més a més es poden afegir tots els senyals que considereu oportuns.



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 3.20 us).

	Valor (hex)
PC	
Instrucció	
Registre 7	
Registre 8	
Registre 9	
Registre 10	
Registre 11	
Registre 12	
Registre 13	
Adreces de memòria	
0x40	
0x44	
0x48	
0x4C	
0x50	
0x54	
0x58	
0x5C	
0x60	

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

Recordeu que s'ha de demostrar el funcionament correcte de la pràctica el dia assenyalar. En aquests cas el dia que s'ha de presentar la pràctica és el 26 d'abril.