LittleProc 2.0. L'evolució als 16 bits, amb arquitectura superscalar i amb pipeline

Annexos

Roc Prat López

28/06/2011

En aquest document es poden trobar els diferents annexos referenciats en la memòria del Projecte Final de Carrera

Taula de Continguts

1 Annex 1: LittleProc de 16 bits ... 3
1.1 UP ... 3
1.2 UC ... 6
1.3 Memòria Principal (RAM) ... 9
1.4 Interrupcions ... 9
1.5 Port d'entrada/sortida ... 11
2 Annex 2: LittleProc de 16 bits segmentat ... 13
2.1 UP ... 13
2.2 UC ... 15
2.3 Memòria Principal (RAM) ... 19
3 Annex 3: LittleProc superescalar amb 2 nuclis ... 21
3.1 UC ... 21
3.2 Memòria Principal (RAM) ... 24
4 Annex 4: Memòria ROM del LittleProc de 16 bits ... fet apart
5 Annex 5: Memòria ROM del LittleProc segmentat ... fet apart
6 Annex 6: Memòria ROM del LittleProc superescalar ... fet apart
Llista de taules ... 27
Llista de figures ... 27

1 Annex 1: LittleProc de 16 bits

Com s'ha vist anteriorment, el LittleProc de 16 bits ha sofert varies variacions respecte el LittleProc original. En termes generals, dir que la principal millora realitzada és que ara el LittleProc podrà tractar dades de 16 bits, mentre que anteriorment només tractava dades de 8 bits. Per a poder realitzar aquesta millora ha sigut necessari augmentar el tamany d'instrucció de 12 a 22 bits. Seguidament es veurà en detall totes les parts del processador:

1.1 UP

Com s'ha explicat en el capítol del LittleProc de 16 bits, el nombre de registres que té la unitat de procés s'ha vist augmentat, passant a tenir-ne 5 més. Ara la UP ens queda de la següent manera:

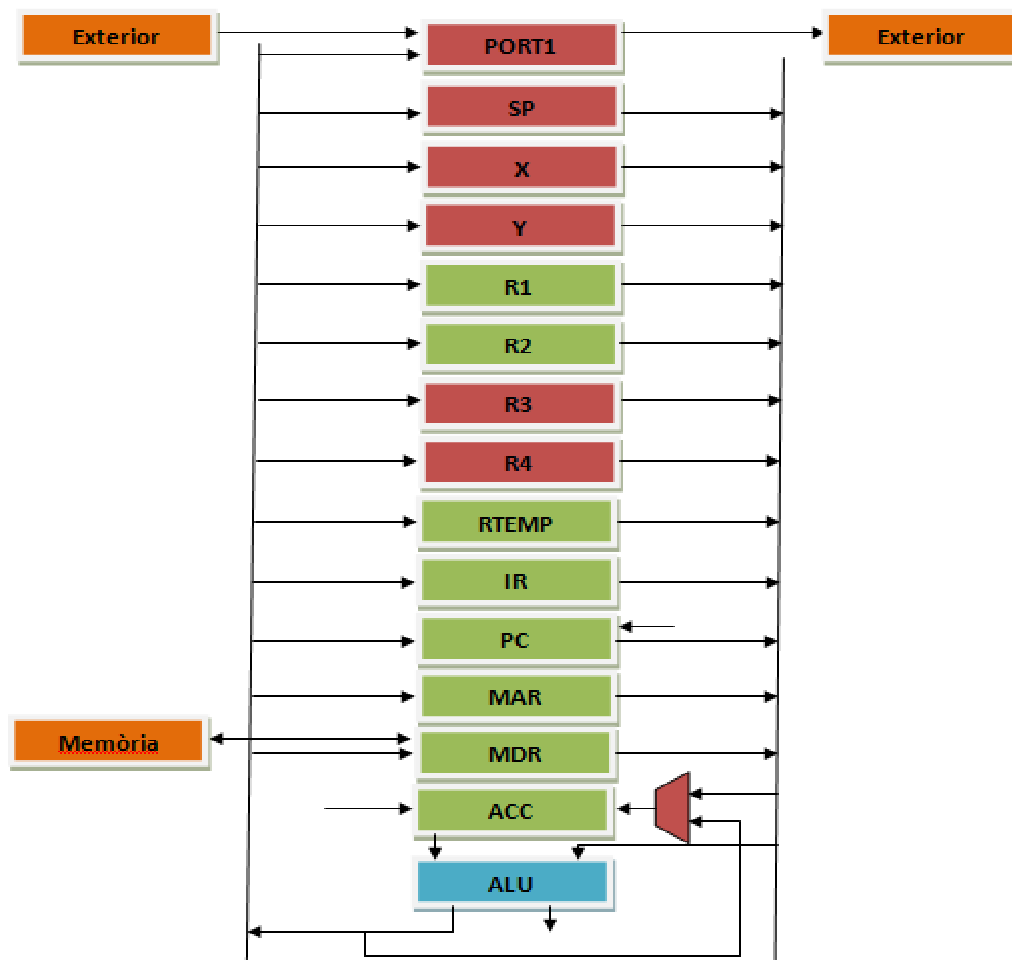

Figura 1.1 UP LittleProc 16 bits

Tal i com podem veure amb color ■ s'han afegit els registres R3, R4, X, Y i SP. Els registres R3 i R4 simplement són per poder emmagatzemar més dades, ja siguin des de memòria o des de el resultat de realitzar les operacions a la ALU. El registre SP s'ha afegit per a poder emmagatzemar el valor del PC quan es produeix una interrupció. Per altre banda, el registre X s'ha afegit per a poder realitzar programes amb bucles while i així poder utilitzar-lo com a contador. Mentre que el registre Y ens servirà per a poder utilitzar-lo com a indexador per poder llegir o escriure dades dintre del bucle.

Comentar també que els tamanys de tots els registres s'han vist augmentats de 12 a 22 bits per a poder satisfer el pas de 8 a 16 bits.

Un altre element que s'ha afegit a la UP és un multiplexor que ens permet seleccionar la entrada del acumulador, ja sigui des de el bus 1, o sigui una dada provinent des de qualsevol registre, o bé des de la sortida de la ALU directament. Això s'ha fet degut a que alhora d'implementar les operacions de llegir o escriure indexadament, es necessitava poder llegir directament el resultat de la màscara de sortida de la ALU i guardar-lo directament al registre acumulador, per així posteriorment poder sumar el resultat al valor Y que teníem i guanyar un cicle de rellotge.

Tal i com s'ha comentat al capítol del LittleProc de 16 bits, a la unitat aritmètica lògica s'han afegit varies operacions noves. Per fer-les s'ha utilitzat els operadors del llenguatge VHDL que ens permeten fer les operacions directament. Aquestes operacions són:

0. Operacions lògiques:
    a. AND: Operació lògica que ens dona el resultat de fer A AND B.
    b. OR: Operació lògica que ens dona el resultat de fer A OR B.
    c. XOR: Operació lògica que ens dona el resultat de fer A XOR B.
    d. NAND: Operació lògica que ens dona el resultat de fer A NAND B.
    e. NOR: Operació lògica que ens dona el resultat de fer A NOR B.
    f. NOT: Operació lògica que ens dona el resultat de fer NOT A o bé de fer NOT B.
1. Operacions aritmètiques:
    a. Multiplicació: Operació aritmètica que mitjançant el operador * ens dona el resultat de multiplicar l'entrada A per l'entrada B.

b. Decrement: Operació que ens dona el resultat de decrementar l'operant B un cop cada vegada.
c. Mascarà de 16 bits: Aquesta operació ens fa una mascara dels 16 últims bits de l'operand. Per fer-ho es fa una AND de l'operand B amb "0000001111111111111111".

Dir també que s'ha afegit una opció a la ALU per a poder assignar el valor 100 a qualsevol registre. Fent això s'evita tenir que assignar-li utilitzant una instrucció. Per altra banda, s'han fet dues operacions aritmètiques més, com són la divisió i el residu. Però com que el llenguatge VHDL no permet realitzar aquestes operacions en binari, donat que no existeix cap operador, s'ha tingut d'utilitzar un petit algorisme explicat seguidament:

```
when "01110" | "01111" => divident := (others => '0');
        for i in 21 downto 0 loop
                divident := divident(21 downto 0)&A(i);
                if (divident < ('0' & B)) then
                        temp(i) <= '0';
                else
                        temp(i) <= '1';
                        divident := (not('0' & B)+1) + divident;
                end if;
        end loop;
        if (Sel = "01111") then
                temp     <=     ("0000000000000000000000"     &
divident(21 downto 0));
        end if;
```

En aquest when hi entra quan el SEL (senyal intern de la ALU que representa el opcode) val "01110" o bé "01111". Per poder realitzar aquestes dues operacions, s'ha declarat una variable anomenada divident de 23 bits inicialment inicialitzada a 0. Un cop entrem dintre del when, ens trobem un for que va de 21 a 0 que serveix per a recórrer els 22 bits del

operand A. Dintre d'aquest for a cada iteració el que es fa és agafar els 22 bits menys significatius del divident i es concatena amb la variable A(i) la posició que li pertoqui. Després es fa una comparació d'aquesta variable amb l'operand B concatenat al principi amb 0. Si la variable divident és més petita, es fica al resultat (senyal temp) un 0 a la posició que li pertoqui (temp(i)). En canvi, si en la comparació la variable divident és més gran, es posa a temp(i) un 1 i a més a més a divident se li assigna el valor de dividir divident per l'operand B, sent aquesta variable a on es guarda el residu. Un cop acabada l'estructura for, tenim en la variable temp emmagatzemat el resultat de la divisió, mentre que en la variable divident hi tenim guardat el residu de la divisió. Per tant si hem seleccionat la opció que es realitzi el mod. Un cop acabada l'estructura for el que es fa és assignar-li a temp el valor del divident.

Comentar que tots els altres registres i busos fan la mateixa funció que feien en el LittleProc.

1.2 UC

En quant a la UC, s'han realitzat diverses variacions per tal de poder controlar tots els canvis realitzats en la UP, donat que ara és té 6 registres nous amb els seus corresponents senyals de Load i de Out. També s'ha tingut d'afegir un bit més de control per a poder seleccionar quina entrada agafar en el multiplexor afegit a la UP com a entrada de l'acumulador. Finalment dir que també s'ha afegit dos bits més en el multiplexor que hi ha dintre de la UC per a seleccionar la condició de negative i la condició interrupció a més a més de les d'inici i zero que teníem en el LittleProc.
Veiem amb més detall cada una de les parts de la UC modificades:

2. Multiplexor de condicions d'entrada:

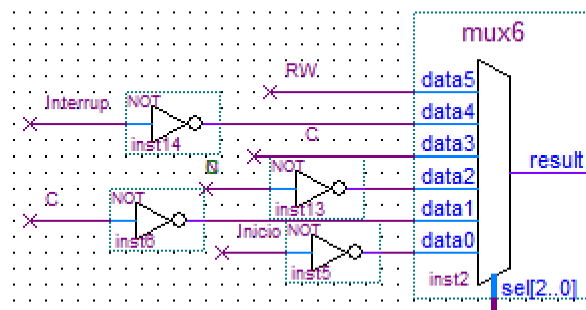

Figura 1.2 Multiplexor de les condicions d'entrada LittleProc 16 bits

Com es pot veure aquest multiplexor té 6 entrades, la de inici negada, la de negative negada, la de zero (C) negada i normal, la de interrupció i finalment també té la d'entrada/sortida. Com que es té 6 entrades, es necessiten tres bits de selecció ja amb 2^2 només podríem tenir 4 entrades i per tant necessitem 2^3. Aquests tres bits de selecció són els bits 42, 41 i 40de la ROM de microinstruccions. Ara mitjançant aquesta modificació, es podrà saltar en cas de que un resultat hagi sigut zero, i per tant es podran crear instruccions noves en el instruction set com són per exemple el BRNEG (salta si és negatiu). També es podrà saltar a la adreça de d'interrupció (assignada per defecte 100) en cas que s'hagi produït una. Més endavant s'explicarà l'ús i control de les interrupcions. Finalment es podrà controlar mitjançant la senyal entrada/sortida si el que es vol és llegir o bé escriure des de l'exterior del processador

3. IRdecoder:

Lògicament, com que s'ha augmentat en número d'instruccions del instruction set, s'ha tingut d'afegir nous casos de decodificació en el IRdecoder, passant de 13 instruccions a decodificar fins a 48 que té el nou instruction set. Com ja sabem, l'opcode són els 6 bits més significatius de cada instrucció. Per tant en el decodificador hi entraran del bit 21 fins al bit16 de cada instrucció. Seguidament es veurà com funciona el IRdecoder:
Suposem que es vol executar la instrucció 25, que en binari és (011001). En el IRdecoder es té un case que selecciona segons l'entrada quina sortida ha de tenir. En aquest cas, si l'entrada és 011001, la sortida la tenim programada en "01011111". Aquesta sortida serà la que s'enviarà al seqüenciador i aquest alhora farà que es comenci a executar la

microinstrucció 95 ("01011111") de la ROM d'instruccions. Aquesta microinstrucció ser la primera de la instrucció a executar. A continuació tenim el instruction set del LittleProc de 16 bits:

Opcode	Instrucció	Paràmetres
0	NOOP	Posició memòria
1	LDR2 EXT	Posició memòria
2	ADD R1, R2	Cap
3	STR1	Posició memòria
4	CMP R1, R2	Cap
5	BRE EXT	Posició memòria
6	JMP EXT	Posició memòria
7	SUBS R1-R2	Cap
8	STR2	Posició memòria
9	FI	Cap
10	LDR1 EXT	Cap
11	SRLR1	Posicions despl.
12	SLLR2	Posicions despl.
13	LDRTemp	Posició memòria
14	SWAP R1, R2	Cap (R1 ⇔ R2)
15	AND R1, R2	Cap
16	OR R1, R2	Cap
17	XOR R1, R2	Cap
18	NAND R1, R2	Cap
19	NOT R1	Cap
20	NOT R2	Cap
21	MUL R1, R2	Cap
22	DIV R1, R2	Cap
23	MOD R1, R2	Cap
24	DEC R1	Cap
25	DEC R2	Cap
26	INC R1	Cap
27	INC R2	Cap
28	MOV R1, RTEMP	Cap
29	MOV R2, RTEMP	Cap
30	STRTEMP	Posició memòria
31	DJNZ X	Posició memòria
32	BRNEG	Posició memòria
33	BRNE	Posició memòria
34	LDR1, Dada	Número
35	LDR2, Dada	Número
36	LDR3 EXT	Posició memòria
37	LDR4 EXT	Posició memòria
38	LDX, Dada	Número
39	LDY, Dada	Número

40	LDR1, Y+num	Posició memòria
41	LDR2, Y+R3	Cap
42	STRTEMP, Y+R4	Cap
43	INC Y	Cap
44	RET	Cap
45	LDRPORT1 EXT	Posició memòria
46	RWPORT1	Dada externa
47	MOV RPORT1, RTEMP	Cap

Taula 1.1 Instruction set LittleProc de 16 bits

4. ROM:

Un altre component modificat en la unitat de control del LittleProc ha sigut la ROM de microinstruccions. Com ja s'ha comentat, a passat a tenir de 25 a 43 bits i una grandària de 64 a 256 posicions donat que lògicament, al implementar més instruccions, s'ha tingut que augmentar el tamany d'aquesta. Per veure com a quedat la ROM de microinstruccions es pot consultar la **Figura**[1].

1.3 Memòria Principal (RAM)

En quant a la memòria principal, que segueix sent compartida per a les dades i per les instruccions, també ha sofert un canvi important, donat que que ara el tamany de les instruccions és de 22 bits envers els 12 bits que tenia el LittleProc. El número de posicions segueix sent el mateix, 128.

1.4 Interrupcions

Una altra de les modificacions introduïdes en el disseny del LittleProc de 16 bits ha sigut poder controlar que passa si es produeix una interrupció. Per a poder implementar-ho s'ha tingut d'afegir un registre més a la UP anomenat SP que com ja s'ha comentat s'encarrega d'emmagatzemar el valor del PC de la instrucció que s'havia d'executar quant s'ha produït la interrupció. Primerament el que s'ha modificat ha sigut el FETCH de la memòria ROM, on si ha afegit tres línies noves, fent que al final d'aquest i abans de que es produeixi el salt cap a la instrucció que s'ha d'executar, es controli si hi ha

Veure figura[1] de les microinstruccions de la ROM a l'annex 4: Memòria ROM del LittleProc de 16 bits

hagut una interrupció. Si és així, s'emmagatzema el valor del PC menys un (es fa així dona que ja s'ha realitzat tot el FETCH, el valor del PC ja ha sigut incrementat per a la següent instrucció, i per tant es perdria una instrucció a executar-se) al SP i es va a la direcció 162 de la memòria ROM, on es posa el PC a 100 (adreça del vector que contindrà l'execució) i és salta a aquella posició de la memòria d'instruccions i dades. També s'ha tingut de dissenyar una instrucció nova anomenada RET que el que fa es retornar un cop acaba la interrupció cap a la execució que estava duent a cau el processador.

En el cas de que no s'hagi produït cap interrupció es continua executant el programa normalment.

Si la interrupció s'ha produït al mig de l'execució d'una instrucció, aleshores no es detecta en el FETCH i per tant no es realitza aquesta.
Seguidament es pot veure com seria l'execució d'un programa on es produeix una interrupció.

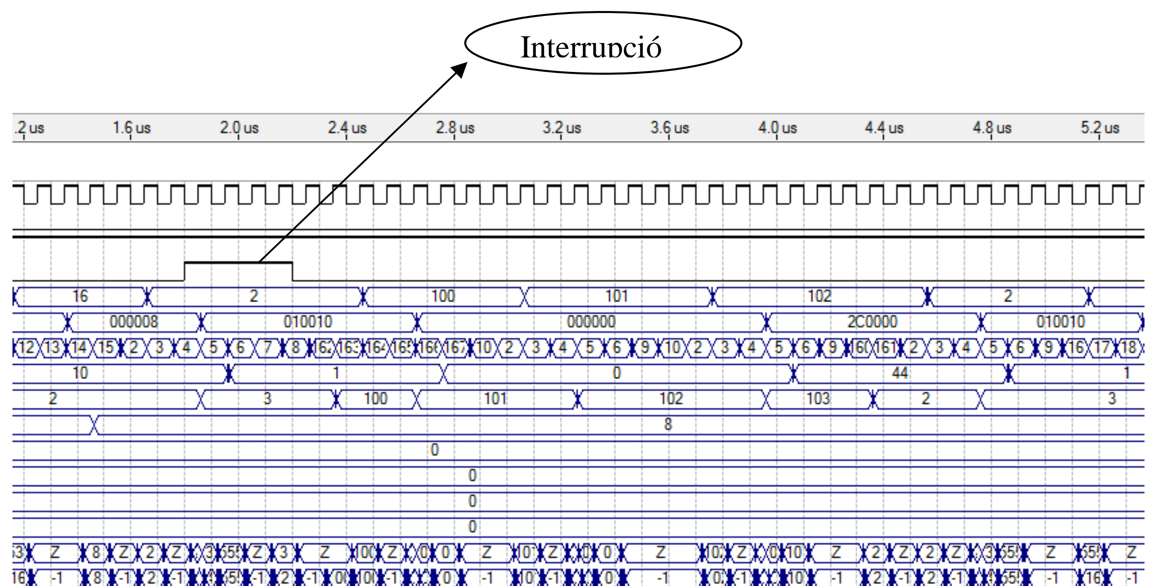

Figura 1.3 Exemple d'interrupció

Tal i com es pot veure a l'instant 1.8 ns entra un interrupció que fa que el programa se'n vagi a executar l'adreça 100 de la RAM (No hi ha res). Fins que a la posició 102 hi ha la instrucció RET que fa retornar al processador al programa que estava executant i seguir amb la seva execució.

1.5 Port d'entrada/sortida

Tal i com s'ha comentat anteriorment, en el LittleProc de 16 bits s'ha afegit un registre anomenat PORT1 mitjançant el qual podem escriure o llegir dades des de l'exterior del processador. Per fer-ho, s'ha afegit dues entrades al processador, la primera de elles és la RW, que és una senyal que si està a 1 el processador escriurà cap a l'exterior, mentre que si està a 0 agafarà la dada de fora. Aquesta senyal és la que va connectada al multiplexor de condicions d'entrada que hem vist anteriorment. Per altra banda, la segona entrada que hem afegit és un bus de 22 bits que serà el que es comunica amb l'exterior. Dins de la UP trobem un nou multiplexor que ens permet seleccionar si carregar la dada externament o bé des de l'interior de la UP.

A més a més de les dues entrades també s'ha afegit una nova sortida que també serà un bus de 22 bits que ens permetrà comunicar amb l'exterior. S'ha tingut de fer servir dos busos separats, un per llegir la dada i un per escriure, ja que el model de la placa no accepta tristates, aleshores els implementa mitjançant portes AND i això ens provoca un error al implementar un bus bidireccional. Si ho implementéssim ens quedaria de la següent manera:

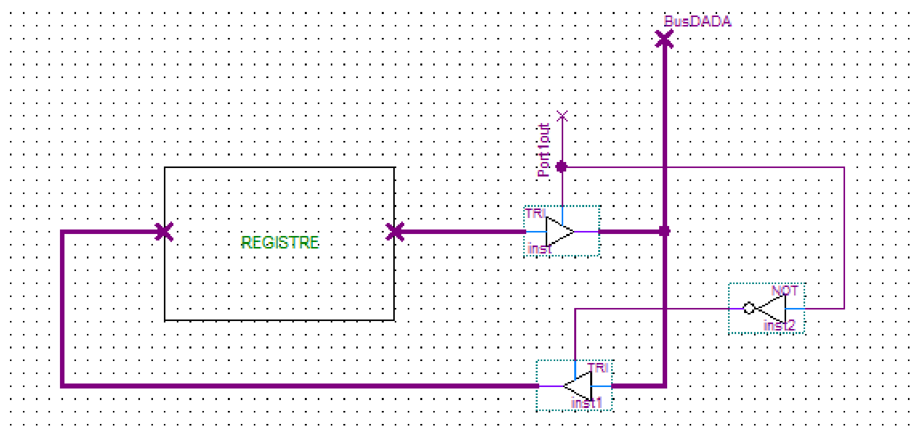

Figura 1.4 E/S en bus bidireccional

Tal com es pot apreciar, si Port1out val 0, mitjançant la porta not ens deixarà carregar una dada al registre, mentre que si Port1out val 1, ens

deixaria escriure cap a l'exterior i com que tenim la porta not, no podríem carregar al registre.

Les 3 noves instruccions afegides per a poder executar l'entrada/sortida són les següents:

1. LDPORT1 EXT: Ens servirà per a carregar qualsevol dada que tinguem en memòria al registre PORT1, així posteriorment es podrà enviar cap a l'exterior.

2. RWPORT1: És la instrucció amb que habilitarem que llegeixi o escrigui a l'exterior, depenent del valor de la senyal RW.

3. MOV RPORT1, RTEMP: Ens servirà per emmagatzemar al registre PORT1 el resultat que haguem obtingut al realitzar una operació aritmètica o lògica.

2 Annex 2: LittleProc de 16 bits segmentat

Com ja sabem, s'ha realitzat una segmentació del processador de 16 bits que havíem realitzat anteriorment per tal de poder augmentar i potenciar el seu rendiment. Per a poder realitzar aquesta segmentació s'ha augmentat en un el número d'unitats aritmètiques lògiques de la nostra unitat de procés, per a poder realitzar dues operacions de càlcul a la vegada. Seguidament es podrà veure com quedat aquesta UP.

2.1 UP

La UP manté els principals registres que la UP del processador que s'havia dissenyat de 16 bits, eliminant només el registre SP que ens servia per a emmagatzemar el PC quan es produïa una interrupció. Per altra banda s'ha afegit un acumulador per la nova ALU. Aquest és el esquema de la nova UP:

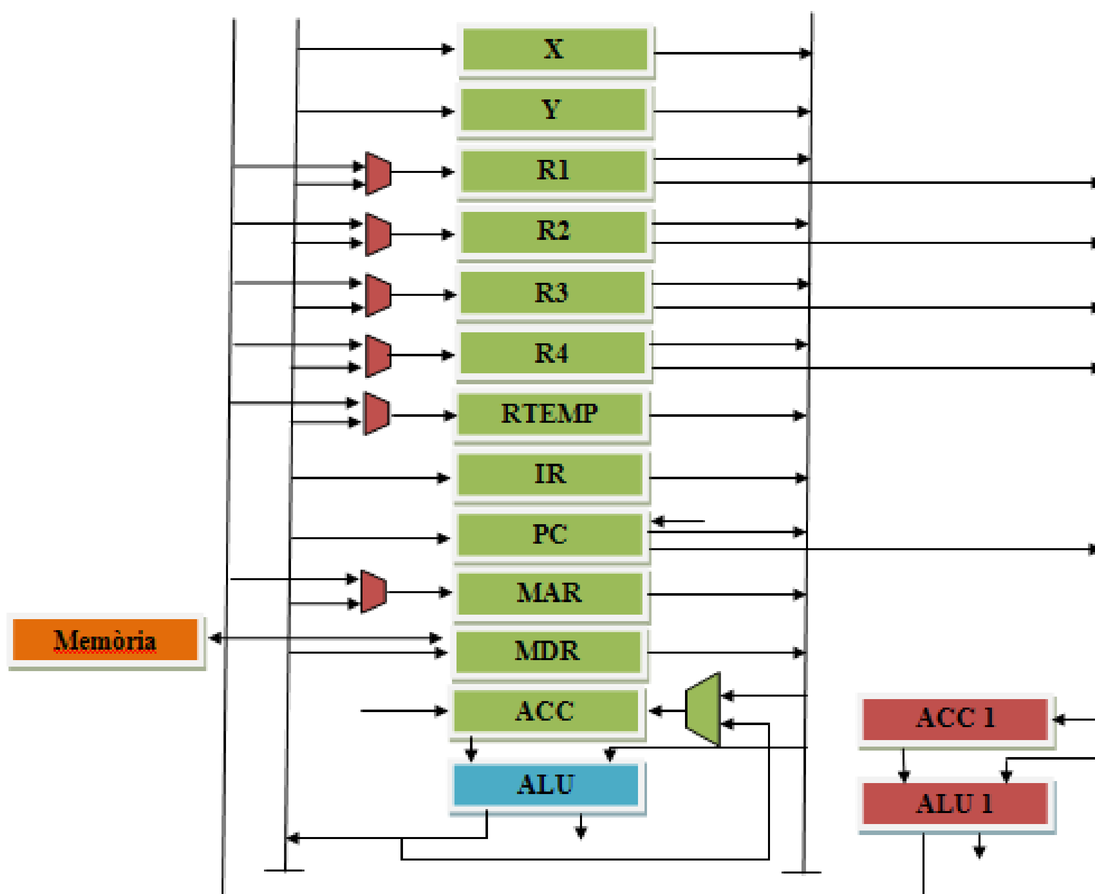

Figura 2.1 UP del LittleProc segmentat

Com es pot apreciar amb color ■ s'ha afegit una nova unitat aritmètica lògica, la ALU 1, per a poder realitzar dues operacions al mateix temps. També es pot veure que s'ha afegit el registre ACC 1 per a poder emmagatzemar un dels operands de la ALU 1. La ALU 1 realitzarà les mateix operacions que la ALU que ja teníem. Aquestes es poden consultar en el annex 1. Per altre banda es pot apreciar que s'ha doblat el número de busos que tenim, passant de dos a quatre. Això s'ha fet per poder proveir de dades a les dues unitats aritmètiques lògiques al mateix temps. Al haver-hi doble bus de sortida dels registres, ara aquests poden o bé enviar una dada per un bus o bé per l'altre (o al dos a la vegada) segons es seleccioni mitjançant les senyals de control d'aquests. Per altra banda, el registres R1, R2, R3, R4, RTemp i MAR, també podran rebre les dades des de un bus o l'altre de sortida de la ALU segons convingui. Per a poder dur a terme aquest fet, s'ha afegit un multiplexor davant de cada registre dels anteriorment esmentats per a seleccionar de quin bus agafar la dada.
Com ja s'ha dit les operacions de les dues ALUs són les mateixes que amb el LittleProc de 16 bits. Aquestes són:

1. Operacions aritmètiques:
    a. Suma
    b. Resta
    c. Mascara de 8 bits
    d. Mascara de 16 bits
    e. Increment i decrement 1
    f. Desplaçament cap a l'esquerra o dreta *n* llocs
    g. Multiplicació
    h. Divisió i Mòdul
2. Operacions lògiques:
    a. AND
    b. OR
    c. XOR
    d. NAND
    e. NOR
    f. NOT

A més a més d'haver-hi el pas transparent de dades a través de les ALUs. Com es pot observar però, en aquest cas no hi ha l'opció d'assignar directament el valor 100 a la sortida de la ALU degut que en aquest cas no cal accedir al vector d'interrupcions perquè en el LittleProc de 16 bits segmentat no es controlen aquestes.

2.2 UC

En aquesta implementació, ha augmentat considerablement el número de bits de la UC degut a que com ja s'ha dit, ara els registres necessiten dos senyals per a controlar per quin bus sortirà la dada. També s'ha afegit els multiplexors davant d'alguns registres per a controlar per quin bus agafar la dada, i per tant també es necessiten nous senyals per a controlar aquests multiplexors. Una altra modificació que ha fet augmentar el número de bits a sigut que al afegir una altra unitat aritmètica lògica, es necessiten 5 bits més nous per a controlar el opcode d'aquesta ALU. Per altra banda el multiplexor de condicions d'entrada en aquest cas s'ha reduït a quatre entrades degut a que ja no es controla si es produeix una interrupció.

1. Multiplexor de condicions d'entrada:

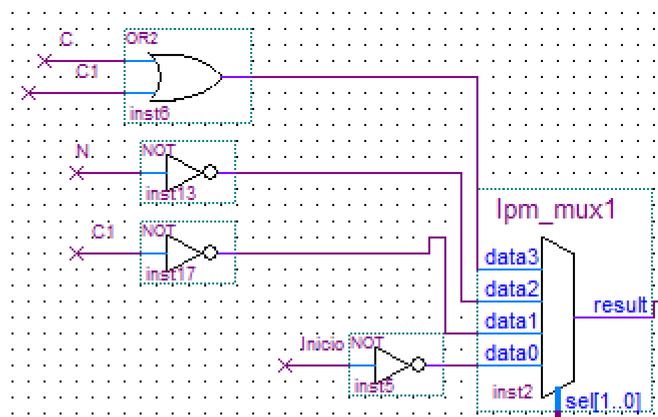

Figura 2.2 Multiplexor de condicions d'entrada del LittleProc segmentat

Es pot observar que ara al tenir dues unitats aritmètiques lògiques, cada una de elles treu una senyal de zero diferent segons l'operació que ha realitzat, i per tant ens pot interessar comprovar si alguna de les dues a tret la senyal de zero. Per això s'ha afegit una porta OR per a comprovar si

alguna de les dues o bé totes dues val zero. Les altres entrades es pot comprovar que són la de negative, la de si no és zero la sortida de la ALU 1 i finalment la de si no Inici.

2. IRdecoder:

En quant al IRdecoder, dir que aquest no ha canviat gaire respecte al del LittleProc de 16 bits, ja que només s'ha eliminat una de les instruccions que podia executar l'altre processador. Aquesta instrucció és la de RET que ens permetia poder tornar des de l'execució d'una interrupció. El instruction set d'aquesta implementació quedarà de la següent manera:

Opcode	Instrucció	Paràmetres
0	NOOP	Posició memòria
1	LDR2 EXT	Posició memòria
2	ADD R1, R2	Cap
3	STR1	Posició memòria
4	CMP R1, R2	Cap
5	BRE EXT	Posició memòria
6	JMP EXT	Posició memòria
7	SUBS R1-R2	Cap
8	STR2	Posició memòria
9	FI	Cap
10	LDR1 EXT	Cap
11	SRLR1	Posicions despl.
12	SLLR2	Posicions despl.
13	LDRTemp	Posició memòria
14	SWAP R1, R2	Cap (R1 ⇔ R2)
15	AND R1, R2	Cap
16	OR R1, R2	Cap
17	XOR R1, R2	Cap
18	NAND R1, R2	Cap
19	NOT R1	Cap
20	NOT R2	Cap
21	MUL R1, R2	Cap
22	DIV R1, R2	Cap
23	MOD R1, R2	Cap
24	DEC R1	Cap
25	DEC R2	Cap
26	INC R1	Cap
27	INC R2	Cap
28	MOV R1, RTEMP	Cap

29	MOV R2, RTEMP	Cap
30	STRTEMP	Posició memòria
31	DJNZ X	Posició memòria
32	BRNEG	Posició memòria
33	BRNE	Posició memòria
34	LDR1, Dada	Número
35	LDR2, Dada	Número
36	LDR3 EXT	Posició memòria
37	LDR4 EXT	Posició memòria
38	LDX, Dada	Número
39	LDY, Dada	Número
40	LDR1, Y+num	Posició memòria
41	LDR2, Y+R3	Cap
42	STRTEMP, Y+R4	Cap
43	INC Y	Cap
44	MOV R3, RTEMP	Cap
45	MOV R4, RTEMP	Cap

Taula 2.1 Instruction set LittleProc segmentat

On es pot veure que hi ha 46 instruccions que pot executar el LittleProc segmentat.

3. ROM:

Un altre component modificat en la unitat de control del LittleProc segmentat ha sigut la ROM de microinstruccions. Com ja s'ha comentat, a passat a tenir de 25 a 56 bits (que representa un augment del 224% del número de bits) i una grandària de 64 a 256 posicions donat que lògicament, al implementar més instruccions, s'ha tingut que augmentar el tamany d'aquesta. Seguidament s'explicarà com s'ha realitzat el segmentat de 3 operacions com son el LDR1, ADD i STR2.

- LDR1:

El programa de control d'aquesta instrucció és el següent:

1. EXE (MAR ← IR) amb mask de 8 bits
2. EXE (NOOP)
3. EXE (MDR MEM(MAR))
4. EXE (R1 ← MDR)
   EXE (MAR ← PC)
5. GOTO 4

On podem veure que les microinstruccions 1, 2 i 3 són les mateixes que en el LittleProc de 16 bits sense segmentar. Però a la microinstrucció 4, podem observar que es fa dues coses. La primera és assignar a R1 el que tenim a MDR, com ja fèiem en el LittleProc de 16 bits, mentre que per altra banda, també assignem el valor del PC al MAR. Això ho podem fer degut a que tenim dues unitats aritmètiques lògiques que ens permeten poder passar dues dades per les dos ALUs i per els dos busos que tenim en cada entrada i cada sortida de la ALU. El pas del valor del PC al registre MAR és la primera microinstrucció del FETCH, i per tant, quan s'ha acabat d'execució d'aquesta microinstrucció, ja es pot accedir a la tercera microinstrucció del FETCH, millorant en dos microinstruccions el temps d'execució.

- DIV:

El programa de control d'aquesta instrucció és el següent:

1. EXE (ACC1 ← R1)
   EXE (MAR ← PC)
2. EXE (ACC1 / R2)
3. EXE (ACC1 / R2)
   EXE (PC ++)
   EXE (MDR MEM(MAR))
4. EXE (RT ← ACC / R2)
   EXE (IR ← MDR)
5. BRA

En aquesta instrucció veurem que encara s'aconsegueix un grau de segmentació més elevat degut a que s'aconsegueix realitzar tota la fase del FETCH al mateix temps que s'executa la instrucció de la divisió. Es pot observar que en la primera microinstrucció i mitjançant les dues ALUs, podem guardar el valor de R1 a l'acumulador 1 i assignar el valor del PC a MAR. En la segona microinstrucció només dividim l'acumulador pel valor de R2 perquè en la fase del FETCH la microinstrucció que hi pertoca és la de NOOP. Posteriorment, a la microinstrucció 3 si que a més a més de dividir el valor de l'acumulador 1 per R2, fem al mateix temps la incrementació del registre PC i assignem al MDR el valor de la memòria de la direcció MAR, que pertoca a la tercera microinstrucció del FETCH. En la següent

microinstrucció d'aquesta instrucció també es fa alhora dues operacions. La primera d'elles és l'assignació del resultat de la divisió al registre RTemp, mentre que l'altre és l'assignació del MDR al registre IR. Com que ja s'ha realitzat tot el FETCH, ara ja es pot saltar directament amb un BRA cap a la següent instrucció a executar-se, evitant així, tota la fase del FETCH que pertocaria.

- STR2:

El programa de controla d'aquesta instrucció és el següent:

1. EXE (MAR ← IR) amb mask de 8 bits
2. EXE (MDR ← R2)
3. EXE (escriptura en RAM)
4. GOTO 2

En aquesta instrucció es pot observar que no hi ha cap segmentació. Això es degut a que sempre es necessita el MAR per a dur-la a terme. Com que no es pot modificar el valor d'aquest registre, mai no es pot realitzar la primera microinstrucció de la fase FETCH que seria la d'assignar al registre MAR el valor del registre PC. Això provoca que quan s'ha acabat d'executar-se aquesta operació s'hagi d'anar mitjançant el GOTO 2 fins a la primer microinstrucció del FETCH.

2.3 Memòria Principal (RAM)

En quant a la memòria principal, en aquesta implementació ha sofert un canvi important, degut a que s'ha canviat l'arquitectura d'aquesta passant de l'arquitectura Von Neumann a l'arquitectura Harvard, tenint ara una memòria de dades independent de la memòria de programa. En quant a les instruccions aquestes segueixen tenint un tamany de 22 bits tai i com tenien en el LittleProc de 16 bits. El número de posicions segueix sent el mateix, 128. Per veure com a quedat la ROM de microinstruccions segmentada es pot consultar la **Figura**[1].

Veure figura[1] de les microinstruccions de la ROM a l'annex 5: Memòria ROM del LittleProc segmentat

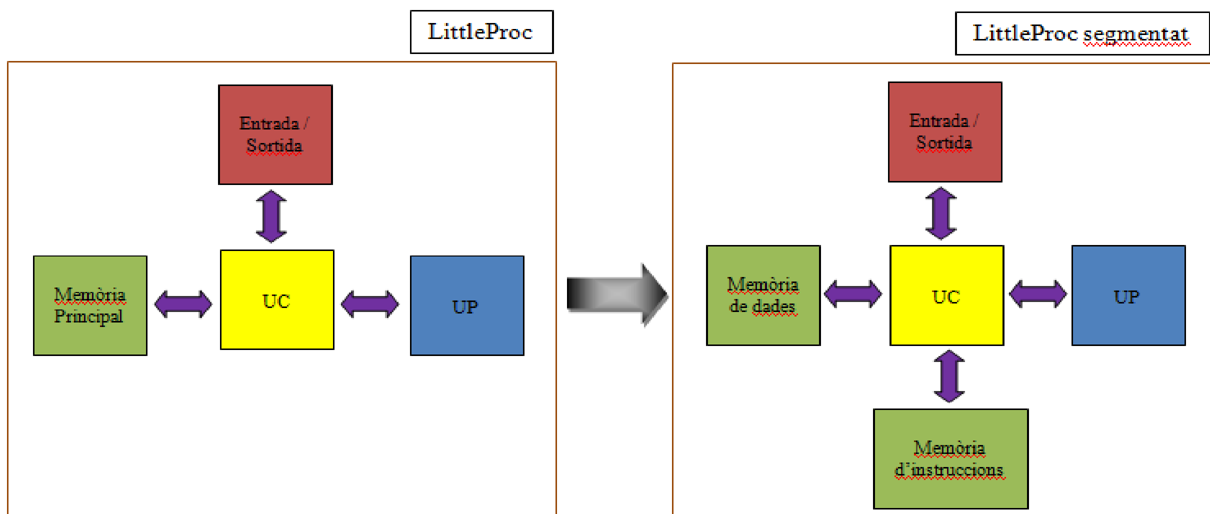

Figura 2.3 Arquitectura Harvard en LittleProc

3 Annex 3: LittleProc superescalar amb 2 nuclis

Tal i com s'ha explicat anteriorment, la última evolució del LittleProc ha sigut convertir-lo en un processador superescalar amb dos nuclis, que ens permet executar dos instruccions al mateix temps, una per cada nucli. Aquesta evolució utilitza una arquitectura Harvard tal i com s'utilitzava en el LittleProc segmentat. S'ha de dir que la UP de tots dos nuclis no ha variat respecte la del LittleProc segmentat, que es pot consultar en el *annex 2*.

3.1 UC

En quant a la unitat de control, dir que tampoc ha canviat el número de bits que utilitza per a controlar tots els senyals de cada component del processador respecte al LittleProc segmentat. Un altre dels elements analitzat anteriorment com és el multiplexor de condicions d'entrada tampoc a canviat. Però per contra si que han canviat els següents elements:

1. IRdecoder:

Tal i com hem dit en el capítol del LittleProc superescalar, el IRdecoder d'aquest ha de poder decodificar 49 instruccions per a cada processador. Això es degut a que s'ha afegit tres instruccions noves per tal de poder dividir en dos parts l'execució d'un programa que consisteixi en realitzar operacions sobre un vector d'un determinat tamany, aconseguint realitzar aquestes operacions amb més de la meitat de temps que ho faria el LittleProc original. Aquestes noves operacions afegides són les següents:

- LDX = R3 / 2:

Aquesta instrucció s'utilitzarà per a dividir entre dos el tamany del vector prèviament carregat al registre R3. Té les següents micoinstruccions:

1. EXE (ACC ←R3)
   EXE (MAR ← PC)
2. EXE (X ← ACC >> R4)

3. GOTO 4

On podem veure que el que es fa és desplaçar un lloc cap a la dreta (és el mateix que dividir entre dos) el que tenim guardat a R3 (número de posicions que té el vector). Aquest 1 (per indicar que es desplaci un lloc) ho tenim guardat a R4.

- LDX = (R3/2) + 1:

  Aquesta instrucció és la mateixa que l'anterior, però ens serviria per a poder realitzar la divisió entre dues parts d'un vector de tamany senar. Per implementar aquesta instrucció s'utilitza les mateixes microinstruccions que les explicades anteriorment, afegint al final una incrementació en 1 del registre X.

- R4 = 2 * R3:

Aquesta instrucció ens servirà per calcular el doble del tamany del vector i així després poder utilitzar el registre R4 indexadamanent per a poder guardar les dades de cada operació que es va realitzant al processador. Les microinstruccions que té aquesta instrucció són les següents:

1. EXE (ACC ← R3)
   EXE (MAR ← PC)
2. EXE (R4 ← ACC << R4)
3. GOTO 4

Per altra banda dir que també s'ha modificat les següents 2 instruccions que teníem respecte del LittleProc segmentat:

LDR2 Y + num → LDR2 Y + R3

Aquesta modificació ens serveix per a poder agafar les dades del segon vector indexadament, només tenint que anar augmentant la Y per a poder canviar la posició.

STRT Y + num → STRT Y + R4

Amb aquesta modificació també podem anar guardant les dades al tercer vector indexadament.

I finalment respecte a la instruccions dir que en el segon processador, la instrucció LDY = num a passat a ser LDY X + num, així fent que el segon processador començy les seves iteracions a la segona meitat del programa.

Per tant el instruction set d'aquests dos processadors queda de la següent manera:

Opcode	Instrucció	Paràmetres
0	LDR1 EXT	Posició memòria
1	LDR2 EXT	Posició memòria
2	ADD R1, R2	Cap
3	STR1	Posició memòria
4	CMP R1, R2	Cap
5	BRE EXT	Posició memòria
6	JMP EXT	Posició memòria
7	SUBS R1-R2	Cap
8	STR2	Posició memòria
9	FI	Cap
10	NOOP	Cap
11	SRLR1	Posicions despl.
12	SLLR2	Posicions despl.
13	LDRTemp	Posició memòria
14	SWAP R1, R2	Cap (R1 ⇔ R2)
15	AND R1, R2	Cap
16	OR R1, R2	Cap
17	XOR R1, R2	Cap
18	NAND R1, R2	Cap
19	NOT R1	Cap
20	NOT R2	Cap
21	MUL R1, R2	Cap
22	DIV R1, R2	Cap
23	MOD R1, R2	Cap
24	DEC R1	Cap
25	DEC R2	Cap
26	INC R1	Cap
27	INC R2	Cap
28	MOV R1, RTEMP	Cap
29	MOV R2, RTEMP	Cap
30	STRTEMP	Posició memòria
31	DJNZ X	Posició memòria
32	BRNEG	Posició memòria
33	BRNE	Posició memòria
34	LDR1, Dada	Número
35	LDR2, Dada	Número

36	LDR3 EXT	Posició memòria
37	LDR4 EXT	Posició memòria
38	LDX, Dada	Número
39	LDY, Dada O bé LDY, X + Dada	Número
40	LDR1, Y+num	Posició memòria
41	LDR2, Y+R3	Cap
42	STRTEMP, Y+R4	Cap
43	INC Y	Cap
44	MOV R3, RTEMP	Cap
45	MOV R4, RTEMP	Cap
46	LDX R3 / 2	Cap
47	LDX (R3 / 2) + 1	Cap
48	LDR4 (2*R3)	Cap

Taula 3.1 Instruction set LittleProc multicore

Per veure com a quedat la ROM de microinstruccions del LittleProc multicore es pot consultar la **Figura**[1].

3.2 Memòria Principal (RAM)

Tal i com es tenia en la anterior evolució del LittleProc, aquesta implementació utilitza una arquitectura Harvard, tenint per separat una memòria de dades i una memòria per les instruccions que ha d'executar el programa. Per altra banda s'ha de dir que ara cadascuna d'aquestes memòries son de doble port. Aquest fet significa que les memòries poden tenir accés simultani a dos llocs diferents de la memòria. Això el que ens permet és que es pugui accedir des de els dos processador a la vegada a la memòria i així per exemple poder guardar al mateix temps els dos resultats de les operacions realitzades. Seguidament es pot observar com seria el símbol d'una memòria de doble port:

Veure figura[1] de les microinstruccions de la ROM a l'annex 6: Memòria ROM del LittleProc superescalar

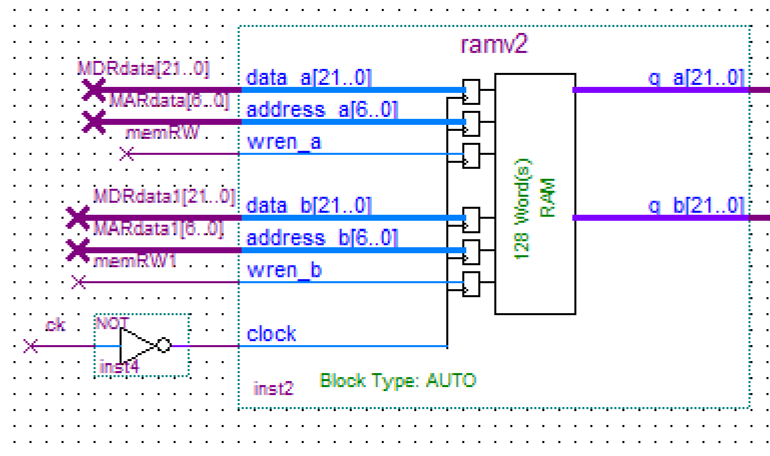

Figura 3.1 Memòria de doble port

En ella poder veure que tenim dos ports d'entrada i dos ports de dos sortida controlats pel mateix rellotge. A més a més es pot observar que també es necessiten dos senyals, una per cada processador, per indicar si s'està fent una lectura o una escriptura en memòria. Cada port d'entrada i de sortida anirà connectat a un processador diferent.

4 Annex 4: Memòria ROM del LittleProc de 16 bits

Fet en un document PDF apart

5 Annex 5: Memòria ROM del LittleProc segmentat

Fet en un document PDF apart

6 Annex 6: Memòria ROM del LittleProc superescalar

Fet en un document PDF apart

Llista de taules

Taula 1.1 Instruction set LittleProc de 16 bits 9
Taula 2.1 Instruction set LittleProc segmentat 17
Taula 3.1 Instruction set LittleProc multicore 24

Llista de figures

Figura 1.1 UP LittleProc 16 bits 3
Figura 1.2 Multiplexor de les condicions d'entrada LittleProc 16 bits 7
Figura 1.3 Exemple d'interrupció 10
Figura 1.4 E/S en bus bidireccional 11
Figura 2.1 UP del LittleProc segmentat 13
Figura 2.2 Multiplexor de condicions d'entrada del LittleProc segmentat 15
Figura 2.3 Arquitectura Harvard en LittleProc 20
Figura 3.1 Memòria de doble port 25