UNIVERSITAT DE BARCELONA

Examen parcial d'Estructura de Computadors (EI) **A**
Novembre de 2019

Nom:
Cognoms :

Optimització de processadors (8 punts)

1. Afegiu tantes instruccions "NOOP" com creieu al codi següent per tal que pugui funcionar en un *pipeline* de 5 etapes com l'emprat al processador de problemes. En aquest cas tenim 32 registres que van des de l'x0 fins a l'x31. (0.5 punts)

Programa	**Execució amb NOOPs**
Add x11, x12, 5	Add x11, x12, 5
Add x13, x11, x12	NOOP
Add x14, x11, 15	NOOP
Add x15, x11, x11	NOOP
	Add x13, x11, x12
	Add x14, x11, 15
	Add x15, x11, x11

Nota: És possible que no necessiteu emprar totes les caselles

	Temps →												
Instrucció	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
Add x11,x12,5	F	D	C	M	W								
Add x13,x11,x12		*	*	*	F	D	C	M	W				
Add x14,x11,15						F	D	C	M	W			
Add x15,x11,x11							F	D	C	M	W		

Nota: ***Aquesta taula és d'ajut, no necessiteu omplir-la. El seu contingut no es valorarà.***

Nom:
Cognoms :

2. La importància de tenir un bon predictor de "braches" depèn de la freqüència amb què s'executen els "branches" condicionals. Si també considerem la precisió del pronòstic dels "branches", tots dos conceptes determinaran la quantitat de temps que el processador es passa aturat a causa de "branches" imprevistos. En aquest exercici, suposem que el desglossament d'instruccions en diverses categories d'instruccions és el següent: (1.5 punts)

ALU/Logic	Branch	Load	Store
40%	25%	25%	10%

Assumeix també les precisions dels predictors de següents:

Always-Taken	Always-Not-Taken
45%	55%

Suposeu que fem servir el processador de 5 etapes emprat a problemes, on els "branches" es resolen al final de l'etapa d'execució. Suposeu, també, que el programa és infinit i partim d'un CPI ideal. Responeu i **argumenteu breument** les següents qüestions:

a. Quin és l'increment en el CPI degut a la mala predicció del predictor "Always-Taken"? (0.5 punts)

En el processador de 5 etapes triguem 2 clocks (brach penalty) en saber si saltem. Això vol dir que al CPI= 1 li hem de sumar el retard dels "penalties" => (0.25)*(1-0.45)*2 (% de branches * % d'equivocacions * cicles que perdem). Així doncs **passem de CPI 1 a 1.275**

b. Quin és l'increment en el CPI degut a la mala predicció del predictor "Always-Not-Taken"? (0.5 punts)

(0.25)*(1-0.55)*2, **passem de CPI 1 a 1.225**

c. L'ús d'una solució dinàmica, com una taula de d'història de salts, milloraria les prestacions ? Compara-ho amb els dos predictors estàtics emprats en aquesta qüestió. (0.5 punts)

Molt bona pregunta. Tenim gairebé un 50% de prob. De saltar (o no saltar) per tant és qüestió d'atzar encertar si el següent salt el farem o no. Tot i així, amb l'històric de salts, com a molt podríem aconseguir el mateix que el always not taken

Nom:
Cognoms :

3. Tenim un processador amb 6 instruccions. La codificació de les 6 instruccions es mostren a la taula següent: (1 punt)

Instruction	Opcode	16-bit encoding				Function
Mov Ra, d	0000	Opcode **(4 bits)**	Destination Register **(4 bits)**	Address **(8 bits)**		RF[a]←M[d]
Mov d, Ra	0001	Opcode **(4 bits)**	Source Register **(4 bits)**	Address **(8 bits)**		M[d]←RF[a]
Add Ra,Rb,Rc	0010	Opcode **(4 bits)**	Destination Register **(4 bits)**	Source Register **(4 bits)**	Source Register **(4 bits)**	RF[a]←RF[b] + RF[c]
Mov Ra, #C	0011	Opcode **(4 bits)**	Destination Register **(4 bits)**	Constant **(8 bits)**		RF[a] ← c
Sub Ra,Rb,Rc	0100	Opcode **(4 bits)**	Destination Register **(4 bits)**	Source Register **(4 bits)**	Source Register **(4 bits)**	RF[a]←RF[b] - RF[c]
Jumpz Ra, X	0101	Opcode **(4 bits)**	Source Register **(4 bits)**	Offset **(8 bits)**		If RF[a] == 0, PC←PC+offset

Quan es fabriquen xips de silici, els defectes dels materials (per exemple, el silici) i els errors de fabricació poden produir circuits defectuosos. Un defecte molt comú és que una línia de senyal es "trenqui" i registri sempre un 0 lògic. A aquest error se l'anomena falta "stuck at 0". Responeu i **argumenteu breument** les següents qüestions:

a. Si en la codificació de les instruccions tenim una falta "stuck at 0" al bit 8, on el bit 0 correspon al bit LSB i el bit 15 correspon al MSB, afectarà la falta al correcte funcionament del processador ? (0.5 punts)

El bit 8 és el LSB del registre de destí/source segons la instrucció. Això implica que mai podrem fer servir registres imparells però el processador pot funcionar amb normalitat

b. Continuant amb les condicions de la qüestió (a), com es veu afectada la instrucció "Add"? (0.25 punts)

El registre destí ha de ser parell

c. Si la falta "stuck at 0" es troba al bit 15, quines instruccions podrà executar el processador ? (0.25 punts)

TOTES, ja que cap OPCODE té l'MSB a 1

4. En un processador amb un *pipeline* de 5 etapes, com el de problemes, examinem com afecta el *pipeline* al temps de cicle del rellotge del processador. Les qüestions d'aquest exercici suposen que cada etapa triga un temps diferent en executar-se, en particular els temps d'execució de cada etapa són els següents: (2.5 punts)

IF	ID	EX	MEM	WB
250 ps	350 ps	150 ps	300 ps	200 ps

Suposeu també que les instruccions executades pel processador es desglossen de la manera següent:

ALU/Logic	Jump/Branch	Load	Store
45%	20%	20%	15%

a. Quin és el temps de cicle de rellotge **mínim** (en ps) en un processador amb *pipeline* i en un sense *pipeline* ? (0.5 punts)
b. Quant temps triga en executar-se una instrucció de tipus Load en un processador amb *pipeline* i en un sense *pipeline* ? (0.5 punts)
c. Si podem dividir una etapa del *pipeline* en dues noves etapes, cadascuna trigarà la meitat de temps en executar-se que l'etapa original, quina etapa dividiríeu i quin és el nou temps de cicle del rellotge del processador? (0.5 punts)
d. Suposant que no hi ha "*stalls*" ni "*hazards*", quina és percentatge d'utilització de la memòria de dades per a les instruccions executades pel processador? (0.5 punts)
e. Suposant que no hi ha "*stalls*" ni "*hazards*", quina és percentatge d'utilització del port d'escriptura dels Registres per a les instruccions executades pel processador? (0.5 punts)

a) Pipeline (anem al ritme de l'etapa més lenta) => clock cycle = 350 ps; Non-pipeline => clock cycle = 1250 ps
b) Load ha de passar per totes les etapes per tant triga 1250 ps en qualsevol processador (no, amb pipeline triga 350*5)
c) Etapa més lenta és ID, per tant dividim ID. Ara l'etapa més lenta és MEM, per tant clock cycle = 300 ps
d) % LOAD + % Store = 35%
e) Totes instruccions ALU ataquen a registres, totes les Load també => 65%

Nom:
Cognoms :

5. Considereu el fragment de codi màquina d'un RISC-V següent: (2.5 punts)

Instrucció	**Què fa ?**
sd x29, 12(x16)	Store double, guarda el contingut d'x29, adreça base a x16 i offset = 12
ld x29, 8(x16)	Load double, carrega un valor a x29, adreça base a x16 i offset = 8
sub x17, x15, x14	Resta de registres, guardem a x17
beqz x17, label	Saltem si el contingut d'x17 és 0
add x30, x11, x14	Suma de registres, guardem a x30
sub x15, x30, x14	Resta de registres, guardem a x15

Nota: tenim 32 registres que van des de l'x0 fins a l'x31

Suposem que modifiquem el *pipeline* de manera que **només tingui una memòria** (que gestiona tant instruccions com dades). En aquest cas, hi haurà un *risc estructural* cada vegada que un programa hagi d'obtenir una instrucció durant el mateix cicle en què una altra instrucció accedeix a les dades.

El processador RISC-V és molt semblant al que hem emprat a les classes de problemes, en aquest cas considerem que no tenim "data forwarding paths". Responeu i **argumenteu breument** les següents qüestions:

d. Completeu la taula següent executant aquest fragment de codi. (0.75 punts)

	Temps →																
Instrucció	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**
sd x29, 12(x16)	F	D	C	M	W												
ld x29, 8(x16)		F	D	C	M	W											
sub x17, x15, x14			F	D	C	M	W										
beqz x17, label				******	******	*	**F**	D	C								
add x30, x11, x14								F	D	C	M	W					
sub x15, x30, x14									*	*	*	F	D	C	M	W	

***** vol dir Stall

****** vol dir que no podem accedir a instruccions ja que la Mem. Està ocupada amb dades

Nota: les instruccions NOOP són instruccions. En particular, el primer dia de classe teòrica vam veure que les podíem anomenar instruccions de miscel·lània. Així doncs, si el compilador afegeix NOOPs, la memòria d'instruccions les emmagatzemarà. Així doncs, al cicles 4 i 5 NO PODEM FER NOOPs i notem el risc estructural

Nota: És possible que no necessiteu emprar totes les caselles

e. És possible reduir el nombre d'stalls/NOOPs deguts a aquest *risc estructural* reordenant el codi? (0.25 punts)

No, cada instrucció ha de passar pel Fetch; per tant, cada accés a DADES a la memòria (només instruccions LOAD/STORE ho fan) provoca que no es pugui fer el Fetch d'una nova instrucció. Reordenar només canvia les instruccions que entren en conflicte

f. És possible reduir el nombre d'stalls/NOOPs deguts a aquest risc estructural afegint NOOPs? (0.25 punts)

No es pot solucionar un RISC ESTRUCTURAL amb NOOPs ja que, fins i tot, els NOOPs són instruccions que també han de passar pel Fetch

g. En un programa on les instruccions executades pel processador es desglossen de la manera següent:

ALU/Logic	Jump/Branch	Load	Store
52%	12%	25%	11%

Quin percentatge d'stalls espereu obtenir en un processador amb aquest risc estructural ? (0.5 punts)

Només Loads i Stores fan stalls, per tant un 36%

h. Completeu la taula següent executant de nou el mateix fragment de codi, considerant que ara Sí tenim els "data forwarding paths" estudiants al processador de problemes (0.75 punts)

	Temps →																				
Instrucció	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**
sd x29, 12(x16)	F	D	C	M	W																
ld x29, 8(x16)		F	D	C	M	W															
sub x17, x15, x14			F	D	C	M	W														
beqz x17, label				**	**	*	**F**	D	C												
add x30, x11, x14								F	D	C	M	W									
sub x15, x30, x14									F	D	C	M	W								

Nota: És possible que no necessiteu emprar totes les caselles

Nom:
Cognoms :

Memòria cau (2 punts)

6. Suposem que tenim un processador amb un CPI base d'1.0, suposant que tots els accessos a memòria arriben a la memòria cau primària i una velocitat de rellotge de 4 GHz. El temps d'accés a la memòria principal és de 100 ns, inclòs tot el temps necessari per al maneig de "*miss*". Suposem que el percentatge de "*miss*" per instrucció a la memòria cau primària sigui del 2%. Quin és l'increment de velocitat del processador si hi afegim una memòria cau secundària que té un temps d'accés de 5 ns per a un "*hit*" o un "*miss*" i és prou gran com per reduir el 0,5% a la memòria principal? (0.5 punts)

   4GHz => clock cycle de 0.25 ns => Miss penalty a MP de 400 clock cycles

   Total CPI = Base CPI + Increment degut a Mem-stalls = 1 + (2% * 400) = 9

   Si tenim dues caches, un miss en la primera pot ser un hit en la segona. Si fem miss a les dues va a MP. El miss penalty de la segona caché és: 5ns/0.25(ns/clock cycle) = 20 clock cycles

   Total CPI = Base CPI + Increment degut a stalls 2ona cache + Increment degut a Mem-stalls = 1 + (2% * 20) + (0.5% *400) = 3.4

7. Considereu una memòria cau amb 64 blocs i una mida de bloc de 16 bytes. Cada direcció de la memòria principal conté un byte. A quin número de bloc s'adreça la direcció 1200 (en decimal)? A quina línia de caché ha d'anar aquest bloc si la caché és tipus mapejat directe?. Quines adreces, en decimal, estan contingudes dins del bloc que heu calculat ? (1 punt, 0.33 punts per qüestió)

a) Posició de mem. Conté 1 B, adreça 0 té LSByte, així doncs adreça 15 té MSByte del bloc. Cada bloc és de 16 bytes, per tant: 1200/16 = 75. L'adreça 1200 apunta al bloc 75.
b) El bloc 75 li toca anar a la línia:75 MOD 64 = 11
c) Les adreces 1200 fins 1215

8. Per convenció, es denomina una memòria cau segons la quantitat de dades que conté (és a dir, una memòria cau de 4 KiB pot contenir 4 KiB de dades); tanmateix, com haureu comprovat a classe, les memòries cache també requereixen SRAM per emmagatzemar metadades com ara etiquetes i bits d' "status". Per a aquest exercici, examinareu com afecta la configuració d'una memòria cau a la quantitat total de SRAM necessària per implementar-la, considerant que a la SRAM només hi guardarem les etiquetes. Per a totes les parts, suposem que les memòries cau són adreçables byte a byte i que les adreces i les paraules són de 64 bits. (adreçables byte a byte vol dir que, si volem, podem accedir a només un byte de la paraula)

   Calculeu el nombre total de bits, provinents de les etiquetes, que es guardaran a l'SRAM si tenim una memòria cau totalment associativa de **32 KiB amb blocs de 2 paraules**. (0.5 punts)

Cada paraula són 8 bytes i a més cada byte és adreçable => B = 3. Cada bloc té 2 paraules = > W = 1. Així doncs, cada bloc té una mida de 16 bytes (2^4)

La caché té 32 KiB = 2^15 bytes de dades. Cada bloc ha d'anar a una línia de la caché, per tant la caché té 2^15 / 2^4 línies.

Tenim 64 bits d'adreça, dels quals 3 es dediquen a B i 1 a W, la resta són per al TAG. Com i = 0 ja que és totalment associativa, al TAG li corresponen 60 bits. Així doncs, la SRAM per a etiquetes ha de guardar 2^11 (línies) * 60 bits de TAG = 122800 bits

Nom:
Cognoms :