Complemento de Arquitectura de Computadoras

Solución Examen Febrero 2016

- **Indique su nombre completo y número de cédula en cada hoja.**
- **Numere todas las hojas e indique el total de hojas en la primera. Escriba las hojas de un solo lado.**
- **Solo se responderán dudas de letras. No se responderán preguntas en los últimos 30 minutos de la prueba.**
- **La prueba es individual y sin material. Duración de la prueba : 3 hs.**
- **Todas las preguntas tienen el mismo puntaje y el mínimo de aprobación es de un 60%.**

Pregunta 1 (8 puntos)

Comente las diferencias entre E/S programada y E/S por interrupciones.

Pregunta 2 (10 puntos)

Explique los algoritmos de reemplazo de bloques FIFO y LRU utilizados en memorias cachés..

Pregunta 3 (12 puntos)

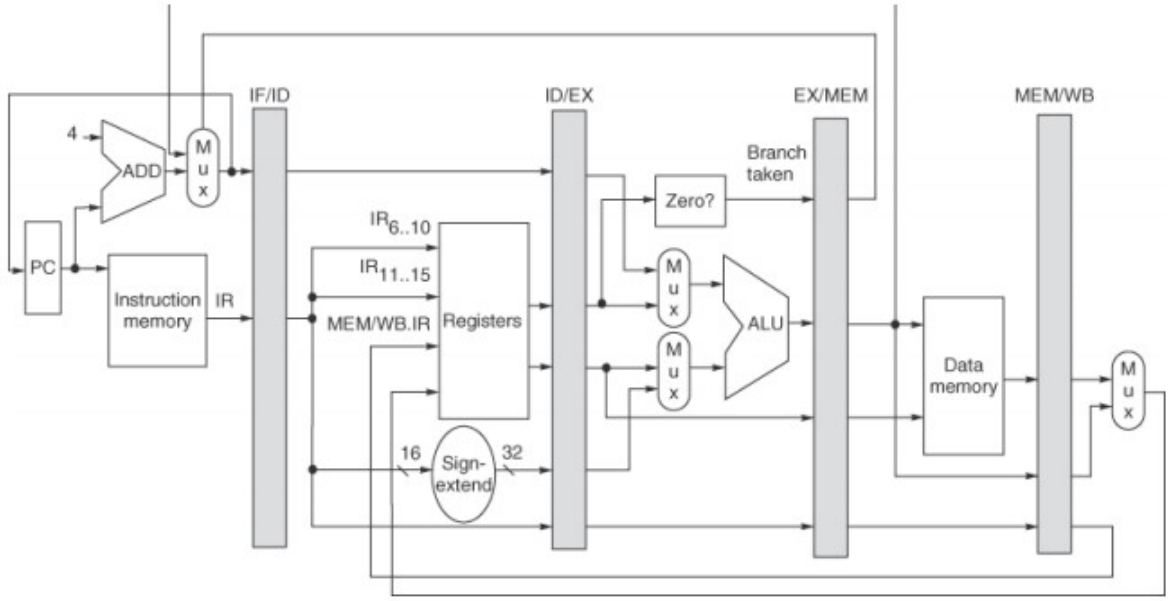
Explique qué es una antidependencia y comente en qué tipo de pipelines se produce. Aporte un ejemplo y comente cómo se puede resolver.

Pregunta 4 (10 puntos)

Deduzca la aceleración ideal de un pipeline de k etapas cuando se ejecutan n instrucciones, con n >> k.

Ejercicio 1 (30 puntos)

Considere un procesador MIPS como el de la figura:



Considere la ejecución del siguiente fragmento de código MIPS:

```
loop:
     lw $3, 400($9)          ; carga en $3 la dirección de memoria
                             ; 400 + $9
     addiu $3, $3, 100       ; $3 = $3 + 100
     sw $3, 400($9)          ; guarda en 400 + $9 el registro $3
     addiu $9, $9, -4        ; $9 = $9 - 4
     bnz $9, loop            ;  si $9 diferente de 0, salta a loop
```

Se pide:

a) ¿Cuál es la penalización por salto tomado en este pipeline? Justifique su respuesta.

b) Dibuje un diagrama del pipeline para la ejecución de una iteración del bucle. Indique el número de ciclos de reloj que toma dicha ejecución. Sabiendo que el valor de $9 al inicio de la ejecución es 4000, calcule el número de ciclos de reloj para la ejecución completa del bucle.

c) Se agrega un predictor de saltos *always taken* en la etapa de decodificación de forma que se reduce a uno el número de ciclos de espera en caso de que la predicción sea correcta. Realice un nuevo diagrama del pipeline para la ejecución de una iteración del bucle. Indique el número de ciclos de reloj demorados en una iteración y en la ejecución total del código.

d) Calcule la aceleración lograda con el agregado del predictor.

Solución:

a) Como se puede apreciar en el dibujo, la señal que indica si el salto es tomado se calcula durante la etapa EX (bloque *zero?*). Durante el ciclo siguiente (cuando el salto está en la etapa MEM), esta señal controla el multiplexor y carga la dirección del salto en el PC. De esta manera, la siguiente instrucción correcta se carga cuando el salto está en la etapa WB, como puede apreciarse en el siguiente diagrama:

Instrucción\Ciclo	1	2	3	4	5	6	7	8	9
bnz $9, loop	IF	ID	EX	MEM	WB				
siguiente instrucción		IF	ID	EX					
siguiente instrucción			IF	ID					
siguiente instrucción				IF					
loop: lw $3, 400($9)		penalización (3 ciclos)			IF	ID	EX	MEM	WB

En total hay 3 ciclos de penalización por salto.

b)

Instrucción\Ciclo	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lw $3, 400($9)	IF	ID	EX	MEM	WB										
addiu $3, $3, 100		IF	ID	ID	ID	EX	MEM	WB							
sw $3, 400($9)			IF	IF	IF	ID	ID	ID	EX	MEM	WB				
addiu $9, $9, -4						IF	IF	IF	ID	EX	MEM	WB			
bnz $9, loop									IF	ID	ID	ID	EX	MEM	WB
lw $3, 400($9)												penalización (3 ciclos)			IF

La primer iteración completa demora **15** ciclos. Luego, como el ciclo 15avo comparte el fin de la ejecución *bnz* y el inicio de la ejecución *lw* de la siguiente iteración, una iteración genérica demora **14** ciclos.

Si $9 = 4000 al inicio de la ejecución, entonces habrán 1000 iteraciones, por lo tanto la ejecución completa demora 1000 * 14 + 1 = 14001 ciclos de reloj.

c)

En este caso, la ejecución de un salto tomado se realiza según el siguiente diagrama:

Instrucción\Ciclo	1	2	3	4	5	6	7
bnz $9, loop	IF	ID	EX	MEM	WB		
siguiente instrucción		IF					
loop: lw $3, 400($9)			IF	ID	EX	MEM	WB

Por lo tanto, la primer iteración sigue demorando 15 ciclos, pero cada siguiente iteración finaliza en **12** ciclos (porque todas las predicciones son correctas). El total de la ejecución demora: 1000 * 12 + 3 = 12003 ciclos de reloj.

d)

Como el ciclo de reloj es igual luego de agregado el predictor, la aceleración se puede calcular como #_ciclos_viejo / #_ciclos_nuevo = 14001 / 12003 = 1.17 (aprox)

Ejercicio 2 (30 puntos)

Dado un conjunto de ***n*** de objetos, una *combinación* C_r^n es una selección de ***r*** objetos sin importar el orden en que se escojan. Las combinaciones se pueden calcular utilizando la siguiente recurrencia:

$$C_r^n = \begin{cases} 1 & si\ r = 0\ o\ r = n \\ n & si\ r = 1 \\ C_r^{n-1} + C_{r-1}^{n-1} & en\ otro\ caso \end{cases}$$

Se pide:

Compile la función ***c*** en 8086, pasando parámetros y resultado en el stack. Deben conservarse los valores de todos los registros y retirarse los parámetros del stack al retornar.

Solución Ejercicio 2

Llamada:

```
push n
push r
call c
pop result
```

Procedimiento

```
c proc
push bp ; guardo el registro bp
mov bp, sp ; apunto con bp al tope del stack
push ax ; conservo registros para luego restaurarlos
push bx
push cx
push dx
mov ax, [bp+6] ; obtengo n
mov bx, [bp+4] ; obtengo r
cmp bx, 0 ; comparo r con 0
je ret1
cmp bx, 1 ; comparo r con 1
je retN
cmp bx, ax ; comparo r con n
je ret1
dec ax ; decremento en uno a n
push ax ; coloco parametros en el stack
push bx ;
call c ; realizo la llamada recursiva con n-1 y r
pop cx ; obtengo el resultado
dec bx ; decremento en uno a r
push ax ; coloco parametros en el stack
push bx
call c ; realizo la llamada recursiva con n-1 y r-1
pop dx ; obtengo resultado
add cx, dx
fin: mov [bp+6], cx ; muevo el resultado a memoria
mov ax, [bp+2] ; obtengo ip
mov [bp+4], ax ; almaceno ip en el lugar correcto del stack
pop dx ; restauro registros
pop cx
pop bx
pop ax
pop bp
```

```
add sp, 2 ; dejo el sp apuntado a ip
ret
ret1: mov cx, 1 ; asigno el resultado del paso base
jmp fin
retN: mov cx, ax ; asigno el resultado del paso base
jmp fin
c endp
```