Parte I. Teoría [36 puntos]
Responda las siguientes preguntas

1- Respecto a las siguientes afirmaciones indique cuales corresponden a elementos de la arquitectura o micro-arquitectura justificando su respuesta (4 puntos)

a- "... está constituido por 32 registros de 128 bits, agrupados en 3 categorías: punto flotante, enteros y vectoriales"
b- "Las 3 etapas de las instrucciones son: *ISSUE*, *Execute* y *Write-Back*"
c- "...Emplea *branch prediction* estadístico mediante un búfer de 128 muestras..."
d- "Los datos almacenados en memoria deben estar alineados con *4-byte alignment*"

2- Explique detalladamente cuál problema se tiene en los sistemas *multi-core* respecto al modelo tradicional de programación. (3 puntos)

3- A continuación se muestra las funciones de 2 sistemas:

$$\text{a-}\ [f(x), g(x), h(x)] = [\frac{(x+1)}{2}, \frac{\sin x}{x}, e^x]$$

$$\text{b-}\ [h(x, y, z)] = [e^{x^2+y^2+z^2}]$$

Según la taxonomía de *Flynn* como se clasifican a y b. Justifique. (4 puntos)

4- Realice un análisis comparativo entre la arquitectura de Harvard modificada y la de von Neuman, listando características, ventajas y desventajas. (5 puntos)

5- Explique detalladamente en qué consiste el direccionamiento relativo al PC. (4 puntos)

6- Explique cómo comúnmente se relacionan las clasificaciones CISC y RISC con LOAD/STORE y REG/MEM. ¿Qué ventajas y desventajas poseen? (4 puntos)

7- En qué consiste el *byte ordering* en un *ISA*, describa cuales tipos hay. (3 puntos)

8- En que consiste la técnica de reordenamiento de código, ¿Cuál es su objetivo? ¿Cómo puede ser evitada? (4 puntos)

9- Discuta sobre la validez de la ley de Amdahl en los sistemas modernos. (5 puntos)

Parte II. Desarrollo [64 puntos]

Resuelva cada uno de los siguientes problemas recuerde indicar todos los pasos que lo llevaron a la solución, además **debe adjuntar su green card/reference sheet** en la cual indique las instrucciones empleadas, con el fin de evaluar su implementación.

1- En la figura 1 se muestra un código en C que corre en una máquina de 32 bits, este emplea una variación del conocido "*Quake 3 C union float trick*", para la manipulación de tipos de datos `float` sin usar la unidad de punto flotante (FPU).

Respecto al código si este corre se le pide lo siguiente:

a- Sabiendo que los `floats` de 4 bytes emplean el estándar IEEE-754, los datos `int` son de 32 bits y los `char` de 1 byte dibuje el contenido en memoria del `union`. ¿Cuánto espacio en bytes consume ésta? (2 puntos)

b- ¿Cuáles son las posibles direcciones de memoria de x[n] si este está naturalmente alineado igual que con los structs, justifique con cálculos el mínimo alineamiento, que sucede con este si en lugar de `float` se emplean tipo `double`? (2 puntos)

c- Que ocurre en la **línea 14** describa cual es la operación aplicada a cada uno de los almacenados en la `x[n]`. Elabore su respuesta. (3 puntos)

d- Implemente el código de las **líneas 13-15** (***for loop***) en asm indicando que ISA va emplear. (8 puntos)

e- Transforme el código de asm desarrollado en 'd' empleando ***loop unrolling*** con un factor de 4. (8 puntos)

f- En qué porcentaje se reduciría la penalización si el *branch predictor* tiene un porcentaje de acierto de 50% y agrega 20 ciclos de reloj al fallar. (3 puntos)

```c
#include <stdio.h>

int main() {
    typedef union {
        float fp;
        int   i;
        char  c;
    } tricky_t;

    //initialise the array of tricky floats x[]
    tricky_t x[8] = {{1.0f}, {-0.5f}, {0.86f} {-1.0f}, {1.0f}, {-0.5f}, {0.86f}, {-1.0f}};

    for(int n = 0; n < 8; n++){
        x[n].i ^= (1 << 31);    // a^=b => a = a^b
    }

    return 0;
}
```

Figura 1. Código C para manipulación de arreglo de `floats`.

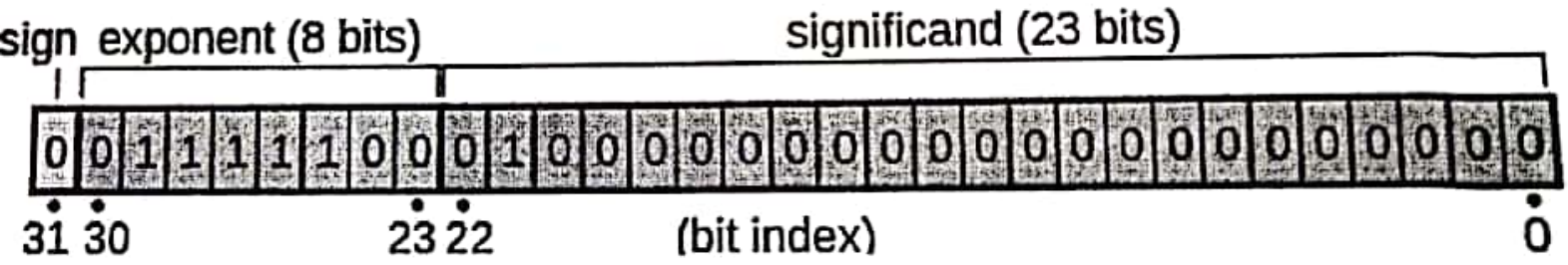


Figura 2. Formato punto flotante IEEE-754

2- En la figura 3 se muestra un código en asm (risc-v)

```
1    loop:
2      slli t3, t0, 2
3      lw  t5, 0(t4)
4      add t4, t1, t3
5      add t5, t2, t2
6      beq t5, x0, loop
```

Figura 3. Código asm risc-v

Respecto al código anterior se le pide lo siguiente

a- Identifique las dependencias presentes. (4 puntos)

b- Clasifique las dependencias encontradas según su tipo, real de datos, anti-dependencia, dependencia de salida.(3 puntos)

c- Si el código anterior no se le aplica OOE, y corre en un pipeline balanceado de 5 etapas, cual es la instrucción a la cual no se le puede aplicar adelantamiento. Justifique (3 puntos)

3- Una técnica empleada para generar señales senoidales digitales consiste en el uso un ***lookup table*** almacenada en memoria que **contiene el primer cuarto** de la onda senoidal $sin(x)$ es decir $x \in [0°, 90°]$ de forma que mediante operaciones de inversión y traslación se puede obtener los valores faltantes es decir se puede completar $sin(x) \forall\, x \in [0°, 360°]$.

Se le pide lo siguiente:

a- Empleando una precisión o tamaño de paso de 1° diseñe el *lookup table*. defina el tipo de datos a usar. ¿Qué tamaño en bytes tiene esta tabla? ¿Cuántas posiciones ocupa el *lookup table* si la memoria es tamaño de palabra de 32 bits? ¿En qué dirección empieza de acuerdo al tipo de dato si se encuentra naturalmente alineado? (6 puntos)

b- Describa el algoritmo que permite obtener la generación de la onda completa a partir de los contenidos del *lookup table* describiendo los tipos de datos, direccionamiento, estructuras de flujo a emplear. (10 puntos)

c- Escriba el código en asm según el ISA que elija tal que pueda generar el valor de la señal

$sin(x)$ para al menos un período empleando el *lookup table* que diseñado en el paso 'a', el valor obtenido en cada muestra debe ser almacenado en memoria. (12 puntos)

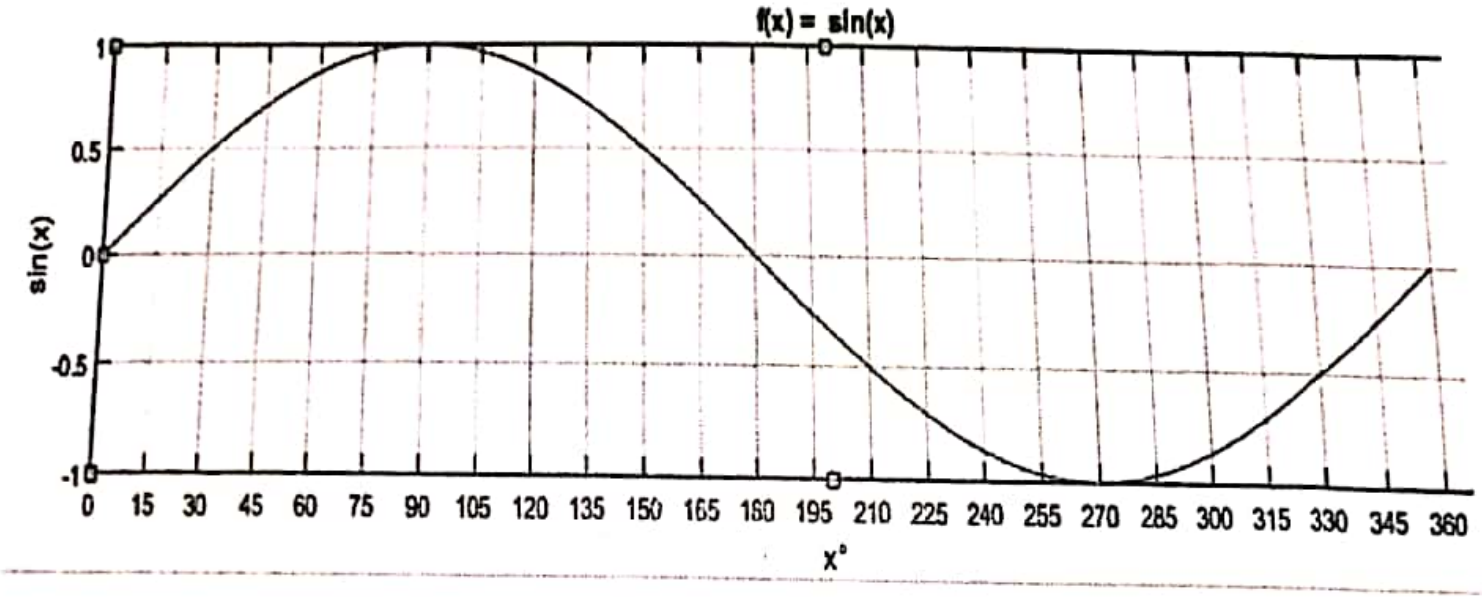


Figura 3. Ejemplo Señal generada a partir del *lookup table* 1/4 $sin(x)$

Preguntas Extra (Escoja 2)

1- Con base en su avance 2 del proyecto grupal I describa cuales características diferenciadores tiene su micro-arquitectura respecto a la base elegida (10 puntos).
2- En que consiste la ley de *Gustafson-Barsis*, como se relaciona con la ley de *Amdahl* (5 puntos)
3- Explique el proceso de encriptación y desencriptación *RSA,* inlcuya las consideraciones respecto a la arquitectura utilizada (10 puntos)
4- En que consiste "*tail call optimasation/tail recursion*" como se diferencia de la recursión "normal" (5 puntos)

Alexis Gavriel Gómez 2016085662 Puntos obtenidos: 64

Examen I · Arquitectura de Computadores

6 de abril, 2019

Ronald Garcia Fernandez

Nota 71,11

28

I Parte

1) Arquitectura / micro arquitectura? 2/4

a) ~~micro~~ arquitectura: no dice explícitamente en la proporción para los registros float, int, vectoriales. solo lo menciona en general.

b) micro arquitectura: nos define cuales son las etapas por las que debe pasar cada instrucción

c) No dice cómo funciona el branch predictor (arquitectura) → da detalles de # muestras y método

d) micro arquitectura: dice cómo está alineada la memoria y sus datos de forma específica ✗ alineamiento es una característica típica de la arch

2) En general no se desarrollaban códigos para ser ejecutados en paralelo, Generalmente los algoritmos se realizan por partes consecutivas. Estaban diseñados para máquinas de 1 core. 3/3

→ Acá pudo mencionar la problemática de sincronización comunicación de los sistema multicore

3) Taxonomía de Flynn

a) $[f(x), g(x), h(x)] = \left[\frac{(x+1)}{2}, \frac{\sin x}{x}, e^x\right]$

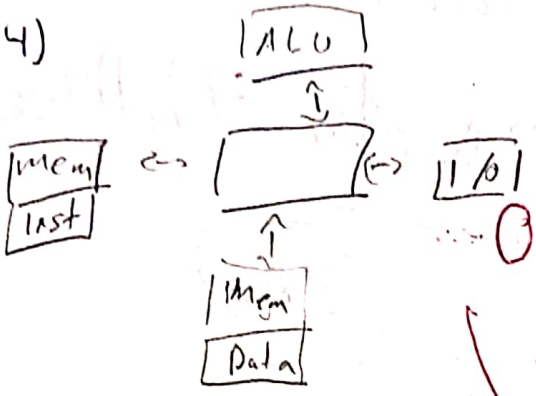
4/4 MISD) sólo existe una variable, x se le aplican varias funciones de forma simultáneas

multiple instructions single data

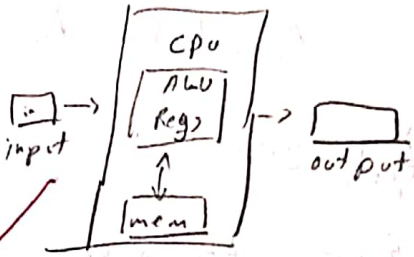
b) $[h(x,y,z)] = [e^{x^2+y^2+z^2}]$

SIMD | Similar al producto punto, existen varios valores x, y, z se aplica una función compleja a todos los datos para obtener 1 valor de respuesta.

4)



0/5



Harvard MODIFICADA

Posee estructuras independientes y bidireccionales

Todo pasa por el CPU

Separa las memorias de Inst y datos

Von Neuman

posee estructuras bidireccionales y unidireccionales

Algunas operaciones pueden evitar usar el CPU.

Una sola memoria, permite modificar instrucciones (metaprogramación)

5) Direccionamiento relativo al PC

0/4

El PC es quien apunta a la direcciones o datos a los utilizar. Se debe actualizar considerando el alineamiento

6) En qué se relacionan CISC - RISC con Load/Store y Reg/Mem

CISC

4/4

se puede accesar a memoria con otras funciones y ser la misma función

Instrucciones más largas y complejas

ADD $R_1, R_2, [R_3]$

"menor cantidad de instrucciones"

RISC

Generalmente solo accesa a memoria con load/store.

Más instrucciones por segundo

ADD $R_1 R_1 R_3$

"mayor cantidad de instrucciones"

byte ordering de un ISA

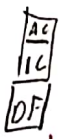
tambien llamado endianness 2,5/3

little endianness

AC 1C DF

big endianness

AC 1C DF



se refiere a en cuales posiciones de memoria se guardan los datos. little: guarda el menos significativo de primero

big: guarda el menos significativo de último

que? menos / mas significativo

g) reordenamiento de código

2/4 General mente hecho por el compilador. no unicamente stalls tambien optimizar uso y ejecucion de unidades funcionales

se busca cambiar el orden de algunas operaciones para evitar stalls en el pipeline. Al reordenar codigo no se debe cambiar la intencion del programador.

reducir dependencias de datos

falta fences / barriers

se puede pedir al compilador no hacer cambios se puede pedir al compilador la no optimizacion

q) ley de Amdahl / en sistemas modernos:

NP	P/N

4/5

Considerando NP como la parte no paralelizable y P/N la paralelizable.

- utilizar muchos cores no siempre es mejor.
- más cores ⇒ más energía, más espacio
- Existen otros limitantes en velocidad, como la memoria
- Generalmente se comparten recursos y datos por lo que puede empeorar el desempeño

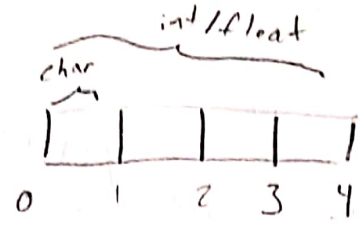
No entiendo si quiere decir + cores - desempeño

en realidad lo que sucede es que nos (3) alcanza el teórico

II Parte

1)

a)

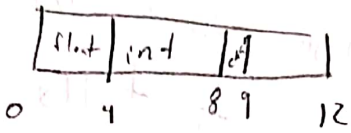


Consume 4 bytes ya que los datos de las uniones usan el mismo espacio en memoria

exc

2|2

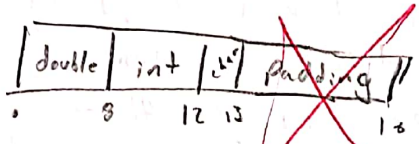
b) S: x[n] está alineado igual que structs:



0|2

se alinea el struct con 12. float int char padding 4 + 4 + 1 + 3 = 12

float → double



?

se alinea con 16 8 + 4 + 1 + 3 padding = 16

c)

1) 1 << 31 ⇒

sign	exp	sig
1	0	0

2) Xor del exp y significando con "0"

casos posibles

e/s	
1	∧ 0 = 1
0	∧ 0 = 0

⇒ los bits de exp y sig quedan igual

3/3

3) Xor signo y "1"

Signo			
0	∧	1	⇒ 1
1	∧	1	⇒ 0

casos

ok

∴ La operación agrega/resta 2^{31} (para int), resta si es mayor, suma si es menor.

∴ La operación en la línea 14 únicamente invierte cambia el signo de cada elemento. (Para float) 4

d) líneas 13 -> 15 en asm

cambiar XOR por EOR

```
SUB  R2,R2,R2          ; contador n : R2
FOR:
     CMP  R2,#8        ; n < 8 ?
     BE   End          ; n == 8 termina
     MOV  R3,#1        ; carga 1 en R3
     LSL  R3,#31       ; corrimiento R3 31 veces
     LSL  R4,R2,#2     ; multiplica por 4 el n
                       ; por el alineamiento en R4
     ADD  R5,R1,R4     ; carga la posición desplazada R5
                       ; R1 guarda el inicio de x
     LDR  R6,R5        ; carga en R6 el valor que apunta R5
     EOR  R7,R6,R3
     STR  R7,R5
     B    FOR
END
```

8/8

e)

```
SUB  R2,R2,R2
FOR:
     CMP  R2,#2
     BE   END
     MOV  R3,#1
     LSL  R3,#31
     LSL  R4,R2,#2
     ADD  R5,R1,R4
     LDR  R6,R5
     XOR  R7,R6,R3
     STR  R7,R5
```

8/10

```
mov R13,R3
mov R14,R4
ADD R14,R14,#4
ADD R15,R1,R14
LDR R16,R15
XOR R17,R16,R13
STR R17,R15

mov R23,R3
mov R24,R4
ADD R24,R24,#8
ADD R25,R1,R24
LDR R26,R25
XOR R27,R26,R23
STR R27,R25
```

copy paste error?

```
mov R33,R3
mov R34,R4
ADD R34,R34,#12
ADD R35,R1,R34
LDR R36,R15
XOR R37,R36,R33
STR R37,R35
B FOR
END
```

y no hay comentarios no indica el bloque basico de codigo

(5)

5)

SO_i^8 → SO_i^2

4 fallos → 1 fallo

80 ciclos → 20 ciclos

0/3

2/

1,5/4

```
1 loop:
2 sll  t3, t0, 2
3 lw   t5, 0(t4)
4 add  t4, t1, t3
5 add  t5, t2, t2
6 beq  t5, x0, loop
```

Dependencias

- lw ∧ R
- w ∧ w
- R ∧ w

→ Por que no uso el # linea no se entiende quien es con quien

Nomenclatura en azul

b) RAW: real de datos, no se tiene el dato hasta escribirlo

wAR: anti dependencia, t4

RAW: salida

0/3

Las que encontro como son

c) la operacion lw en t5 va a obtener el dato hasta el final

3/3

3/

a) 4 bytes float × 9 palabras

3 64 bytes

12 palabras = 11 palabra (352) + 1 palabra (12 bytes)

6/6

?

b)

if $x < 91$

está en la tabla
buscar x en tabla

10/10

elif $90 < x < 181$

$i = 90 - (x - 90) \Rightarrow i = 180 - x$

buscar i en tabla

elif $180 < x < 271$

buscar $(180 - x)$ en tabla

multiplicar resultado por -1

elif $270 < x < 361$

buscar $(270 - x)$ en tabla

multiplicar por -1

c)

11 0/12

Opcional

1/5

4) Se guarda en un registro parte del resultado a calcular, el resultado ~~ka~~ que se llena hasta la iteración actual ~~da~~ Al converger retorna el valor y no ~~regresa~~ necesita regresar al llamado anterior.