Politecnico di Milano – Sede di Cremona
Anno Accademico 2016/2017

Architettura dei Calcolatori e Sistemi Operativi

Esame – 15.09.2017

Prof. Carlo Brandolese

Cognome _____ **Nome** _____

Matricola _____ **Firma** _____

Istruzioni

1. Scrivere con cura, negli spazi sopra segnati, il proprio cognome, nome, numero di matricola e apporre la firma.
2. È vietato consultare libri, eserciziari, appu nti ed utilizzare la calcolatrice e qualunque strumento elettronico (inclusi i cellulari), pena l'invalidazione del compito.
3. Il testo, debitamente compilato, deve essere riconsegnato in ogni caso.
4. Il tempo della prova è di 3 ore

Valutazione

Domanda	Voto	Note
A		
B		
C		
D		
E		
F		

Domanda A

Si consideri il codice seguente:

```
        .data
str:    .asciiz  "Prova di stringa bella"

        .text
main:
        la       $a0,str
        li       $a1,97
        li       $a2,65
        jal      strrep
        la       $a0,str
        li       $v0,4
        syscall
        li       $v0,10
        syscall
```

In cui la funzione strrep ha il seguente prototipo C:

```
void strrep( char* str, char old, char new );
```

e sostituisce nella stringa **str** ogni occorrenza del carattere **old** con il carattere **new**. Si sviluppi tale funzione in assembly, in modo che possa essere chiamata in modo corretto dal codice sopra riportato

Domanda B

Scrivere un programma C secondo le seguenti specifiche:

- Il processo padre accetta sulla linea di comando un argomento numerico intero N che deve essere minore di 16, altrimenti il programma termina.
- Il processo padre crea quindi N processi figli
- Ogni processo figlio genera un numero casuale da 1 a 10 e, se tale numero è maggiore di 5, invia un segnale al processo padre, quindi termina sempre con stato di uscita pari a 0.
- Il processo padre conta il numero di segnali ricevuti e lo stampa a video

Si riportano qui sotto i prototipi di alcune funzioni che potrebbero essere utili nello svolgimento dell'esercizio:

```
pid_t fork(void);
pid_t wait(int *status);
pid_t waitpid(pid_t pid, int *status, int options);
int kill(pid_t pid, int sig);
int raise(int sig);
typedef void (*sighandler_t)(int);
sighandler_t signal(int signum, sighandler_t handler);
void exit(int status);
unsigned int alarm(unsigned int seconds);
```

Si ricorda inoltre che i segnali utente disponibili sono **SIG_USR1** e **SIG_USR2**.

Domanda C

Si consideri un sistema con uno spazio di indirizzamento di 64Kbyte dotato di una cache diretta della dimensione complessiva di 512 byte, organizzata a linee di 32 byte.

1. Si rappresenti l'architettura a blocchi di una tale memoria cache

2. Si calcoli il numero totale esatto di bit necessari a realizzare una tale memoria cache

3. Si indichi la dimensione dei campi tag, index ed offset

   TAG: INDEX: OFFSET:

4. Data la sequenza di accessi riportata di seguito, si completi la tabella indicando se si è avuto un hit, i campi offset, index e tag ed i due bit di valid e dirty, tutti espressi in binario. Si supponga che la cache sia inizialmente vuota.

R/W	ADDR	HIT	TAG	INDEX	OFFSET	V	D
W	0x6E6E						
W	0x6E81						
R	0x6E62						
W	0x386D						
R	0x08D3						
R	0x0090						
R	0x6E64						

Domanda D

Un sistema dotato di memoria virtuale con paginazione e segmentazione di tipo UNIX è caratterizzato dai parametri seguenti:

Memoria logica: 32 Kbyte
Indirizzo fisico: 32 Kbyte
Dimensione pagina: 4 Kbyte

Si considerino tre programmi X, Y e Z caratterizzati dalla seguente dimensione iniziale dei segmenti (C: Code, D: Data, P: Pila)

CX: 12 KB DX: 4 KB PX: 4 KB
CY: 8 KB DY: 8 KB PY: 4 KB
CZ: 4 KB DZ: 8 KB PZ: 4 KB

Completare la seguente tabella, riportando la struttura in pagine della memoria virtuale dei due programmi X e Y sapendo che le pagine vengono allocate sequenzialmente.

Indirizzo Pagina Virtuale	Programma X	Programma Y	Programma Z
0			
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			

Ad un certo istante T_0 le seguenti operazioni sono state completate:

1. Creazione del processo P e lancio del programma Y (“fork” di P ed “exec” di Y)
2. Creazione del processo Q e lancio del programma X (“fork” di Q ed “exec” di X)
3. Accesso in lettura ai dati all’indirizzo 0x23E0 da parte di P
4. Creazione di una nuova pagina di pila da parte di P
5. Creazione del processo R come figlio di P (“fork” eseguita da P)
6. Creazione di una pagina di pila da parte di R
7. Accesso in scrittura ai dati all’indirizzo 0x2F1B da parte di R
8. Terminazione del processo P (exit)
9. Esecuzione della funzione “exec Z” (lancio di Z) nel processo Q e conseguente trasformazione di Q in processo S

Sapendo che:

1. L'esecuzione di un programma avviene caricando inizialmente e in quest'ordine:
   a. La pagina di codice con l'istruzione di partenza
   b. Una pagina di pila
2. Il caricamento di ulteriori pagine in memoria avviene su richiesta (on demand)
3. Il numero di pagine residenti di R è pari a 3
4. L'indirizzo esadecimale di partenza di X è 0x235C
5. L'indirizzo esadecimale di partenza di Y è 0x15D1
6. L'indirizzo esadecimale di partenza di Z è 0x07C4
7. Per la sostituzione delle pagine di memoria si utilizza una politica LRU. Almeno una pagina di pila deve sempre rimanere in memoria
8. L'allocazione delle pagine virtuali nelle pagine fisiche avviene sempre in sequenza
9. All'inizio della sequenza di eventi la MMU è vuota

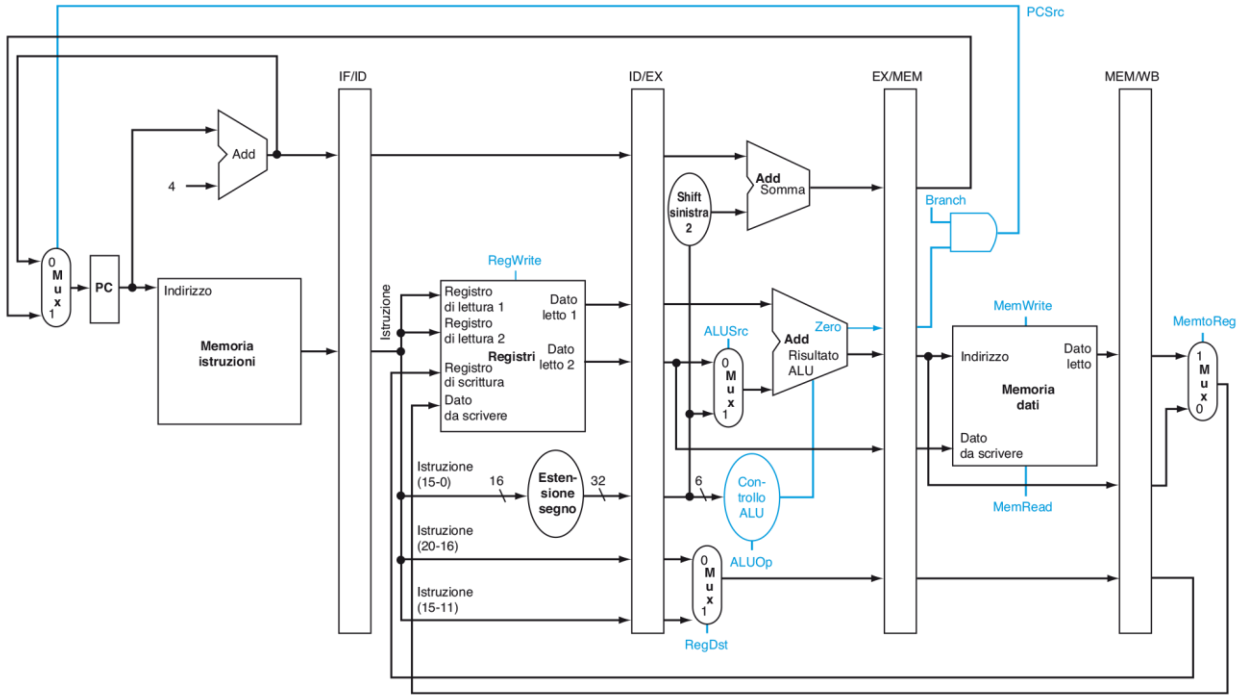
Si riporti nelle tabelle seguenti una descrizione dello stato della memoria fisica e della MMU al termine della sequenza di operazioni di cui sopra.

Memoria Fisica	
Ind. Fisico	**Pag. Allocate**
0	
1	
2	
3	
4	
5	
6	
7	

MMU			
Processo	**NPV**	**NPF**	**Valido**

Domanda E

Sia dato un processore MIPS dotato di una pipeline standard a cinque stadi BASELINE come quello riportato nella figura di riferimento qui sotto.



Sia inoltre dato il seguente codice assembler:

```
1                addi  $t0, $zero, 5
2                addi  $t1, $zero, 2
3                mul   $t1, $t1, $t1
4                sge   $t2, $t0, $t1
5                bnez  $t2, else
6                sub   $t0, $t0, $t1
7                j     endif
8    else:       add   $t0, $t0, $t1
9    endif:      sw    $t0, A
```

1. Si esegua il codice inserendo stalli ove necessario, e si calcoli il numero di cicli di clock necessari per eseguire il codice, e il CPI

N° Istr.	**Cicli di Clock**																													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
ADDI1																														
ADDI2																														
MUL3																														
SGE4																														
BNEZ 5																														
ADD8																														
SW9																														

Numero cicli di clock:

$CPI =$

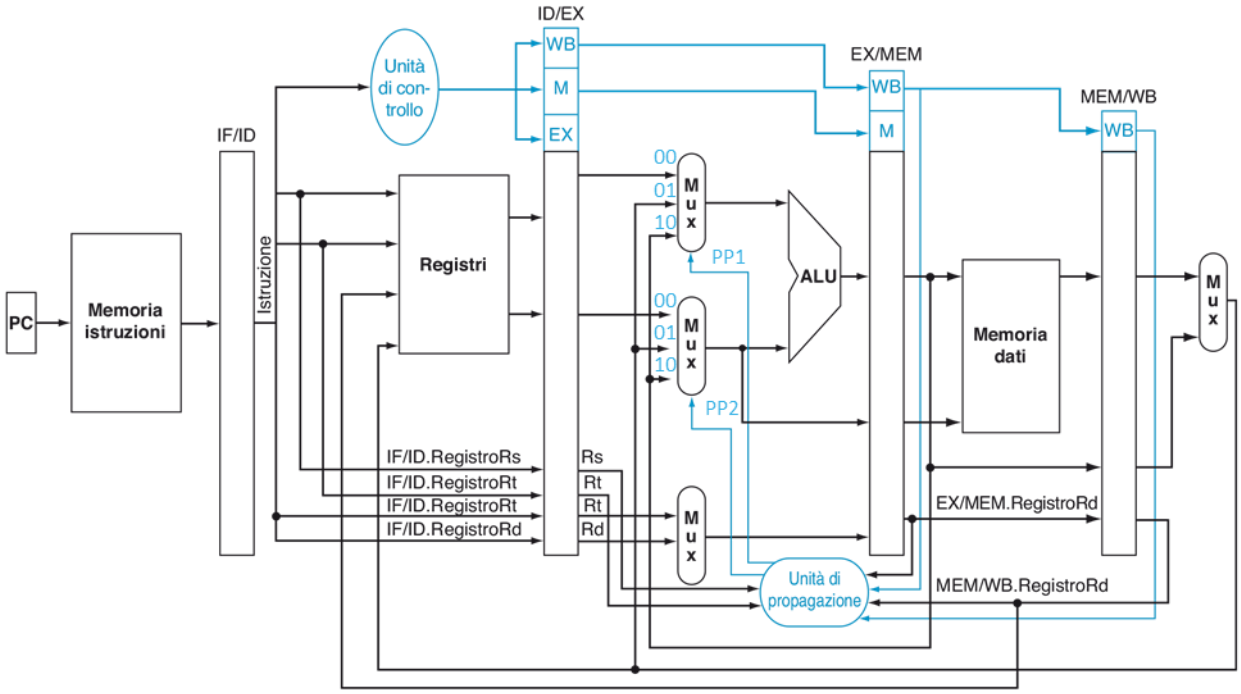
2. Si supponga ora di migliorare l'architettura inserendo i PERCORSI DI PROPAGAZIONE e la PREDIZIONE STATICA BRANCH NOT TAKEN e VALUTAZIONE ANTICIPATA DEL SALTO. Si esegua nuovamente il codice inserendo stalli ove necessario, e si calcoli il numero di cicli di clock necessari per eseguire il codice, il CPI.

N° Istr.	**Cicli di Clock**																													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
ADDI1																														
ADDI2																														
MUL3																														
SGE4																														
BNEZ 5																														
SUB6																														
ADD8																														
SW9																														

Numero cicli di clock:

$CPI =$

3. Si indichino inoltre, data l'architettura ottimizzata, quali sono i valori dei segnali di controllo indicarti in figura (e anche quelli indicati nella figura precedente e non più riportati qui) nel caso dell'esecuzione dell'istruzione: **`mul $t1, $t1, $t1`**



Segnale	Valore
RegDst	
Jump	
Branch	
MemRead	
MemToReg	
ALUOp (operazione)	
MemWrite	
ALUSrc	
RegWrite	
PP1	
PP2	

Domanda F

Si descrivano in modo chiaro e completo i vari passi del flusso di compilazione, inclusi gli aspetti relativi alla costruzione ed all'uso di librerie statiche.