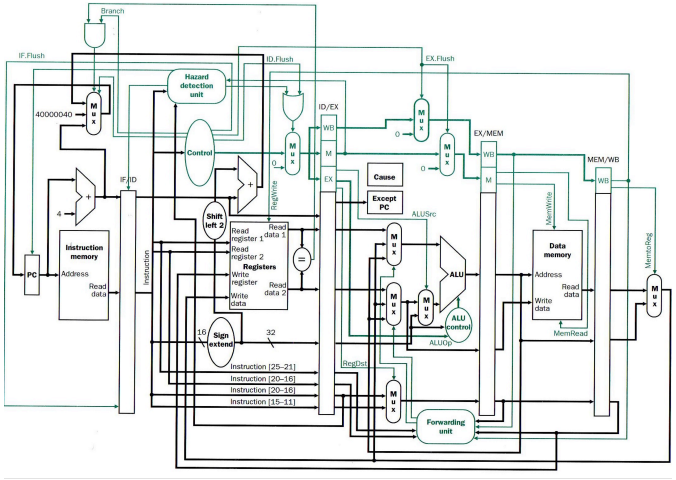
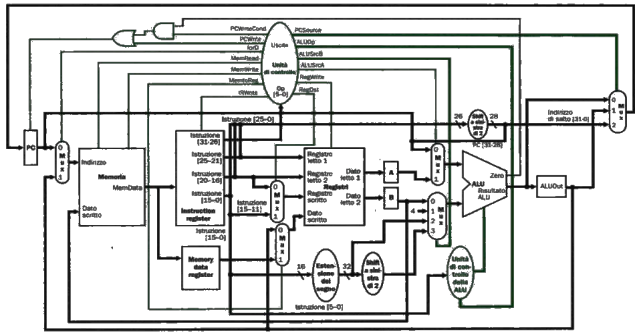
1. [9] Si consideri la CPU seguente, la quale stia eseguendo il codice a lato. a) Identificare e descrivere gli eventuali casi di criticità nel codice;
b) determinare il contenuto del registro **ID/EX** (in uscita) dopo **6 cicli di clock** dall'inizio della prima istruzione;
c) determinare quanti cicli di clock in totale sono necessari per completare tutte le istruzioni, supponendo che i registri **$4** e **$5** contengano valori diversi.

```
0x240:  add $1, $2, $3
        sw $1, 0($2)
        beq $4, $5, -20
        lw $6, 0($5)
        addi $6, $6, 1
```



2. [6] Considerando la CPU seguente, la quale esegua il codice dell'esercizio precedente:
a) identificare e descrivere gli eventuali casi di criticità nel codice;
b) determinare il contenuto dei registri **A**, **B** e **ALUout** (in uscita) dopo **11 cicli di clock** dall'inizio della prima istruzione;
c) determinare quanti cicli di clock in totale sono necessari per completare tutte le istruzioni, supponendo che **beq** non salti.



3. [8] Si traduca in linguaggio Assembly MIPS nativo, evitando cioè di utilizzare pseudo-istruzioni, la seguente coppia di procedure in linguaggio C. Si consideri che entrambe le procedure si aspettano l'argomento nel registro **$a0** e restituiscono il risultato nel registro **$v0**.

```
int LaFunz(int n)
{
 if( n<10 )
    return( Quad4(n) );
 else
    return(Quad4(n)*LaFunz(n/2));
}
```

```
int Quad4(int n)
{
   return( 4*n*n );
}
```

4. [7] Rappresentare il contenuto, byte per byte in formato esadecimale, e gli indirizzi corrispondenti, della zona di memoria che viene modificata a seguito dell'esecuzione del frammento di codice a lato (si ricorda che il codice ASCII di "A"=65).

Determinare il valore contenuto nel registro **$v0** al ritorno dalla chiamata **syscall** contenuta nel programma.

```
        .data 0x500
        .byte –5, +5
        .word –20, 0x20
        .asciiz "ABCDE"
        .space 6
        .text
start:  addi $a0, $zero, 768
        addi $v0, $zero, 9
        syscall
        ...
```

System calls

	codice ($v0)	argomenti	risultato
print_int	1	$a0	
print_float	2	$f12	
print_double	3	$f12	
print_string	4	$a0	
read_int	5		$v0
read_float	6		$f0
read_double	7		$f0
read_string	8	$a0,$a1	
sbrk	9	$a0	$v0
exit	10		

Registri MIPS

0	zero		24-25	t8 – t9
1	at		26-27	k0 – k1
2-3	v0 – v1		28	Gp
4-7	a0 – a3		29	Sp
8-15	t0 – t7		30	s8
16-23	s0 – s7		31	Ra