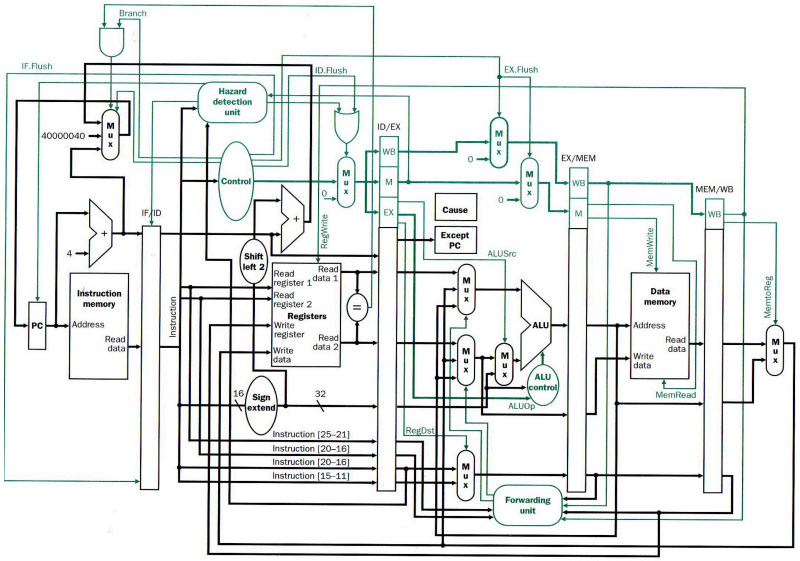
1. [9] Si consideri la CPU seguente, la quale stia eseguendo il codice a lato.

```
0x8F0: lw $1, 0($1)
       sw $1, 0($1)
       lw $6, 0($5)
       add $7, $5, $6
       addi $5, $5, 1
```

a) Identificare gli eventuali casi di criticità nel codice e descrivere come ciascuno viene gestito dalla CPU;
b) tracciare il diagramma delle fasi si esecuzione di ogni istruzione, per ogni ciclo di clock;
c) determinare quanti cicli di clock sono necessari per completare tutte le istruzioni.
d) determinare il contenuto del registro **ID/EX** (in uscita) dopo **6 cicli di clock** dall'inizio della prima istruzione.



2. [8] Si traduca in linguaggio Assembly MIPS nativo, evitando cioè di utilizzare pseudo-istruzioni (ad eccezione dell'istruzione "load address", permessa), la seguente procedura in linguaggio C. L'array ARR va inserito nel segmento dati statici. La procedura si aspetta l'argomento nel registro **$a0** e restituisce il risultato nel registro **$v0**.

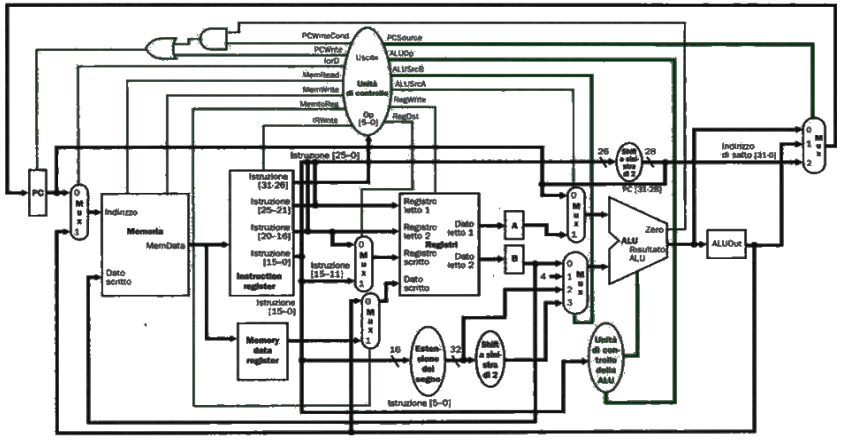
```
int ARR[10] = {1, 1, 2, 3, 5, 8, 13, 21, 34, 55};

int StraFunz(unsigned int n)
{
     if( n>0 )
          return( ARR[mod(n,10)] * StraFunz(n-1) );
     else
          return( ARR[n] );
}
```


3. [7] Si consideri la CPU seguente, mentre esegue l'istruzione:

```
0xA00:  beq $5, $8, -40
```

Determinare (quando possibile) il contenuto dei registri **PC, IR, MDR, A**, **B** e **ALUout** al termine di ogni ciclo di clock di esecuzione dell'istruzione, supponendo che **$5=12** e **$8=0**.



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 restituito dalla prima chiamata alla prima system call **sbrk** contenuta nel programma.

```
.data 0x03F0
.byte –5, +5
.align 2
.asciiz "ALFA"
.align 3
.word –20, 0x20
.space 24
.text
...
```

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