M1 SESI – Architecture des Processeurs et Optimisation

TD2 – Pipeline MIPS

Ce TD se focalise sur les points suivants :

- Le schéma d'exécution des instructions dans la réalisation pipeline du Mips
- La dépendance de données dans le pipeline
- Le problème des branchements dans le pipeline
- Évaluation du nombre cycles nécessaires pour l'exécution d'un code, CPI, CPI-utile

Exercice 1 :

Montrer à l'aide d'un schéma détaillé l'exécution de l'instruction `SLLV rd, rt, rs` (Shift Left Logic Variable) dans le pipeline du Mips-32.

Exercice 2 :

Montrer à l'aide d'un schéma détaillé l'exécution de l'instruction `BLTZAL rs, label` (Branch if Less Than Zero And Link) dans le pipeline du Mips-32.

Exercice 3 :

On considère une nouvelle instruction de branchement conditionnel : l'instruction `BEQPI` (Branch if equal and post increment).

```
    Beqpi    rs, rt, Label
```

Il s'agit d'une instruction de format I. Cette instruction réalise l'opération suivante. Comme pour BEQ, le contenu de RS est comparé au contenu de RT. En cas d'égalité, on se branche à l'instruction portant l'étiquette Label, sinon le branchement échoue et on continue en séquence. De plus, une fois le test effectué, le contenu de RT est incrémenté.

Le schéma d'exécution du pipeline du Mips convient-il à l'exécution de cette instruction ? Si oui, proposer un schéma détaillé montrant l'exécution de cette instruction. Si non, expliquer pourquoi.

Exercice 4 :

On considère une nouvelle instruction de branchement conditionnel : l'instruction BEQPD (Branch if equal and pre-decrement).

```
    Beqpd    rs, rt, Label
```

Il s'agit d'une instruction de format I. Cette instruction réalise l'opération suivante. Comme pour `BEQ`, le contenu de RS est comparé au contenu de RT. En cas d'égalité, on se branche à l'instruction portant l'étiquette Label, sinon le branchement échoue et on continue en séquence. En plus, avant d'effectuer le test, le contenu de RT est decrémenté.

Le schéma d'exécution du pipeline du Mips convient-il à l'exécution de cette instruction ? Si oui, proposer un schéma détaillé montrant l'exécution de cette instruction. Si non, expliquer pourquoi.

Exercice 5 :

Montrer à l'aide d'un schéma détaillé l'exécution de la suite d'instructions suivante dans le pipeline du Mips-32.

```
    Add    r3 , r2 , r1
    Add    r3 , r3 , r1
```

Quelle est la condition du déclenchement des bypass ?

Exercice 6 :

Montrer à l'aide d'un schéma détaillé l'exécution de la suite d'instructions suivante dans le pipeline du Mips-32.

```
    Add    r0 , r2 , r1
    Add    r3 , r0 , r1
```

Donner l'expression exacte de la condition du déclenchement des bypass ?

Exercice 7 :

Montrer à l'aide d'un schéma détaillé l'exécution de la suite d'instructions suivante dans le pipeline du Mips-32.

```
    Lw     r3 , 0 (r2)
    Add    r3 , r3 , r1
```

Exercice 8 :

Analyser l'exécution de la suite d'instructions suivante dans le processeur Mips-32 à l'aide d'un schéma simplifié.

```
    Addiu  r2, r0 , 4
    Lui    r3, 0x000c
    Add    r2 , r2, r2
    Ori    r3, r3, 0x4568
    Lw     r2, 0(r3)
    Lbu    r2, 0(r2)
    Ori    r2, r2, 0x0001
    Bltzal r2, suite
    Addu   r0 , r0 , r0
suite:
    Jr     r31
    Addu   r31, r31, -8
```

Exercice 9 :

La réalisation du processeur Mips-32 nécessite la mise en œuvre de 8 bypass (raccourcis). Pour chacun de ces bypass, proposer un exemple de suite d'instructions qui illustre la nécessité de ce bypass.

Exercice 10 :

La fonction strlen calcule le nombre de caractères :

```
unsigned int strlen (char * str) {
   unsigned int i = 0;
   while (str [i] != '\0') {
      i++;
   }
   return (i);
}
```

Différents codes assembleur peuvent être produits pour cette fonction pour un processeur non-pipeline :

1ère version :

```
_strlen:
    Addiu    r29, r29, -1*4
    Addu     r2, r0 , r0     ; initialisation
_strlen_loop:
    Lb       r9 , 0(r4)      ; lire 1 caractere
    Beq      r9 , r0 , _strlen_end_loop    ; fin de chaine ?
    Addiu    r4 , r4 , 1     ; caractere suivant
    Addiu    r2 , r2 , 1     ; caractere suivant
    J        _strlen_loop    ; boucle

_strlen_end_loop:
    Addiu    r29, r29, 1*4
    Jr       r31
```

NOP →

2ème version :

```
_strlen:
    Addiu    r29, r29, -1*4
    Addiu    r2, r0 , -1     ; initialisation
_strlen_loop:
    Lb       r9 , 0(r4)      ; lire 1 caractere
    Addiu    r4 , r4 , 1     ; caractere suivant
    Addiu    r2 , r2 , 1     ; caractere suivant
    Bne      r9 , r0 , _strlen_loop    ; fin de chaine ?

    Addiu    r29, r29, 1*4
    Jr       r31
```

3ème version :

```
_strlen:
    Addiu    r29, r29, -1*4
    Addiu    r8, r4 ,  1     ; sauvegarde adresse debut
_strlen_loop:
    Lb       r9 , 0(r4)      ; lire 1 caractere (octet)
    Addiu    r4 , r4 , 1     ; caractere suivant
    Bne      r9 , r0 , _strlen_loop    ; fin de chaine ?
    Subu     r2 , r4 , r8    ; calculer nbr de carac

    Addiu    r29, r29, 1*4
    Jr       r31
```

Pour chacune des trois versions, transformer le code de telle façon qu'il puisse être exécuté sur le processeur Mips pipeline à 5 étages.

Analyser l'exécution de la boucle sur le processeur Mips à l'aide d'un schéma simplifié.

Calculer le nombre de cycles nécessaires à l'exécution d'une itération de la boucle.

Calculer le CPI et le CPI-utile de la boucle.

Exercice 11 :

La fonction strupper transforme une chaîne de caractères en majuscules :

```
char * strupper (char * src) {
   int i = 0;
   while (src[i] != '\0') {
      if ((src[i] >= 'a') && (src[i] <= 'z')) {
         src[i] = src[i] - 'a' + 'A'
      }
      i++;
   }
```

```
   return src;
}
```

La compilation de cette fonction pour un processeur non-pipeline a produit le code suivant:

```
_strupper:
    Addiu    r29, r29, -1*4

    Addu     r2 , r0 , r4      ; valeur de retour
    Addiu    r11, r0 , 'a'     ; pour la comparaison
    Addiu    r12, r0 , 'z'     ; pour la comparaison

_strupper_loop:
    Lb       r8 , 0(r4)        ; lire src[i]
    Slt      r9 , r8 , r11     ; src[i] < 'a'
    Slt      r10, r12, r8      ; 'z'    < src[i]
    Or       r10, r10, r9      ; et des 2 conditions
    Bne      r10, r0 , _strupper_endif    ; si 1 des 2 cond vraie
    Addiu    r8 , r8 , 'A'-'a'; transformer en majuscule
    Sb       r8 , 0(r4)

_strupper_endif:
    Addiu    r4 , r4 , 1       ; caractère suivant
    Bne      r8 , r0 , _strupper_loop

    Addiu    r29, r29, 1*4
    Jr       r31
```

Transformer ce code de telle façon qu'il puisse être exécuté sur le processeur Mips pipeline à 5 étages.

Analyser l'exécution de la boucle sur le processeur Mips à l'aide d'un schéma simplifié dans le cas où le `if` (du code source) réussit, puis dans le cas où le `if` échoue.

Calculer le nombre de cycles nécessaires à l'exécution d'une itération de la boucle dans le cas ou le `if` réussit.

Calculer le nombre de cycles nécessaires à l'exécution d'une itération de la boucle dans le cas ou le `if` échoue.

Sachant que 30% des caractères sont minuscules calculer le CPI et le CPI-utile de la boucle.

Exercice 12 :

La fonction suivante calcule le PGCD de deux nombres positifs non nuls :

```
unsigned int pgcd (unsigned int a, unsigned int b) {
   while (a != b) {
      if (a < b) {
          b = b - a;
      }
      else {
          a = a - b;
      }
   }
   return a;
}
```

La compilation de cette fonction pour un processeur non-pipeline a produit le code suivant:

```
_pgcd:
    Beq      r4 , r5 , _pgcd_end
```

```
_pgcd_loop:
    Sltu    r8 , r4 , r5    ; a < b
    Beq     r8 , r0 , _pgcd_else
    Subu    r5 , r5 , r4
    J       _pgcd_endif

_pgcd_else:
    Subu    r4 , r4 , r5
_pgcd_endif:
    Bne     r4 , r5 , _pgcd_loop

_pgcd_end:
    Add     r2 , r4 , r0    ; valeur de retour
    Jr      r31
```

Transformer ce code de telle façon qu'il puisse être exécuté sur le processeur Mips pipeline à 5 étages.

Analyser l'exécution de la boucle sur le processeur Mips à l'aide d'un schéma simplifié, dans le cas où le `if` (du code source) réussit, puis dans le cas où le `if` échoue.

Calculer le nombre de cycles nécessaires à l'exécution d'une itération de la boucle dans le cas ou le `if` réussit.

Calculer le nombre de cycles nécessaires à l'exécution d'une itération de la boucle dans le cas ou le `if` échoue.

Sachant que dans un cas sur deux le `if` échoue, calculer le CPI et le CPI-utile de la boucle.

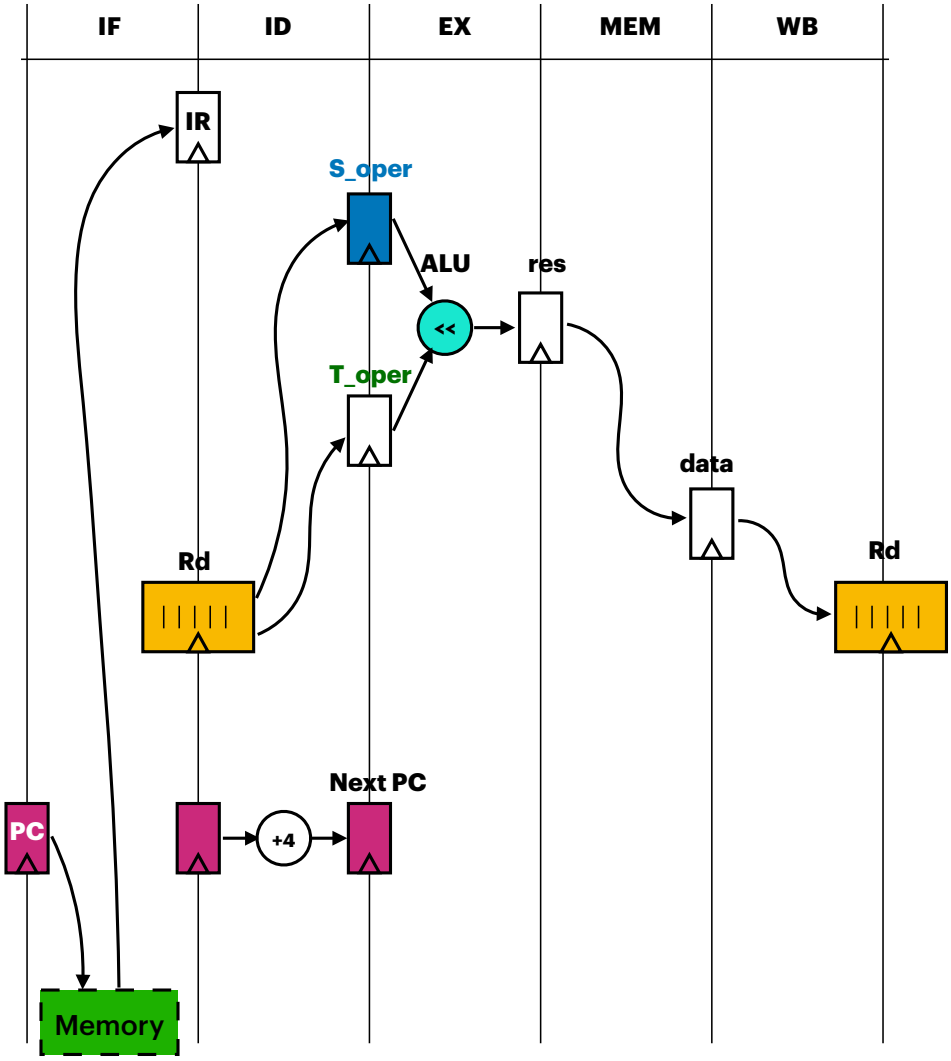
Last modified on Oct 19, 2018, 10:50:54 AM

Exercise 1

Instruction

SLLV Rd, Rt, Rs

1. Fetch the instruction from Memory
2. Load the operands
3. Execute the instruction (ALU)
4.



Exercise 2

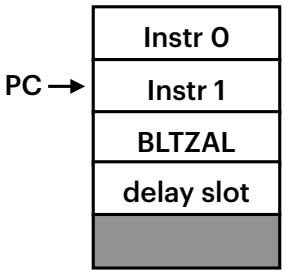
Instruction

BLTZAL Rs, label

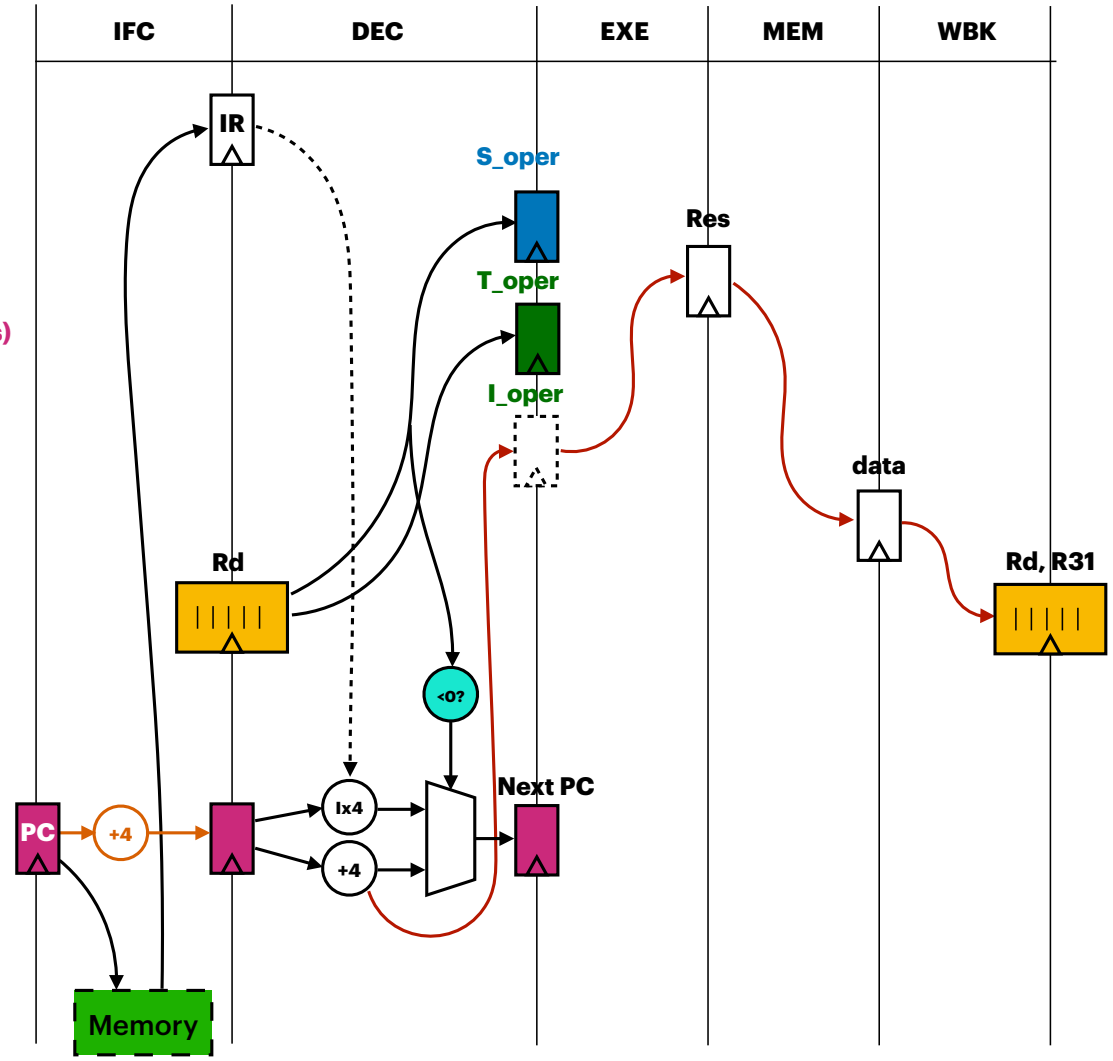
Lesser than zero

Link - Write R31 (return address)

Rt = Label (Immediate)



Since we're adding +4 before the multiplexer, we don't have to add it again



Exercise 3

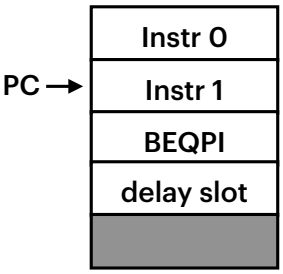
Instruction

BEQPI Rs, label

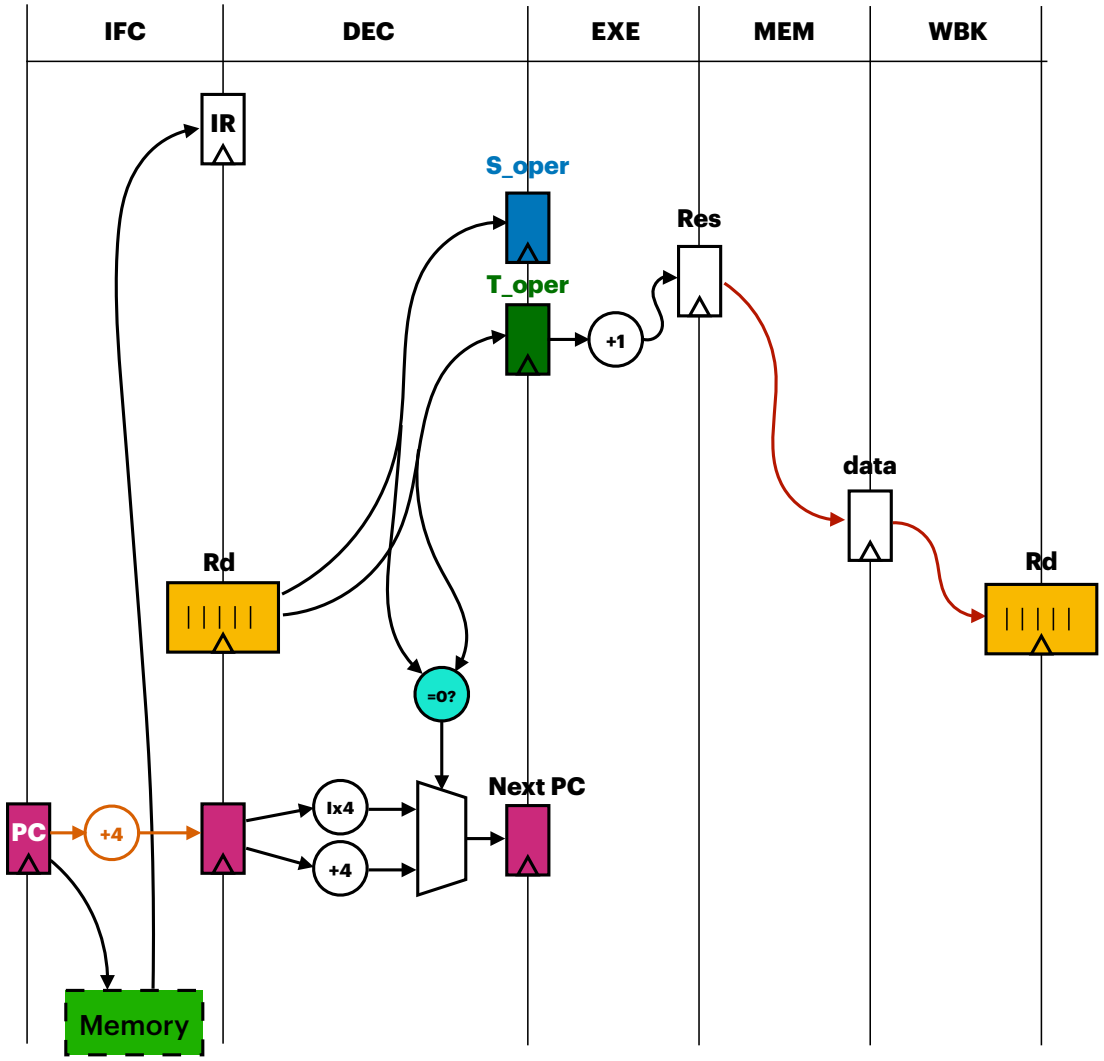
Equal zero

Post Increment

Rt = Label (Immediate)



Since we're adding +4 before the multiplexer, we don't have to add it again

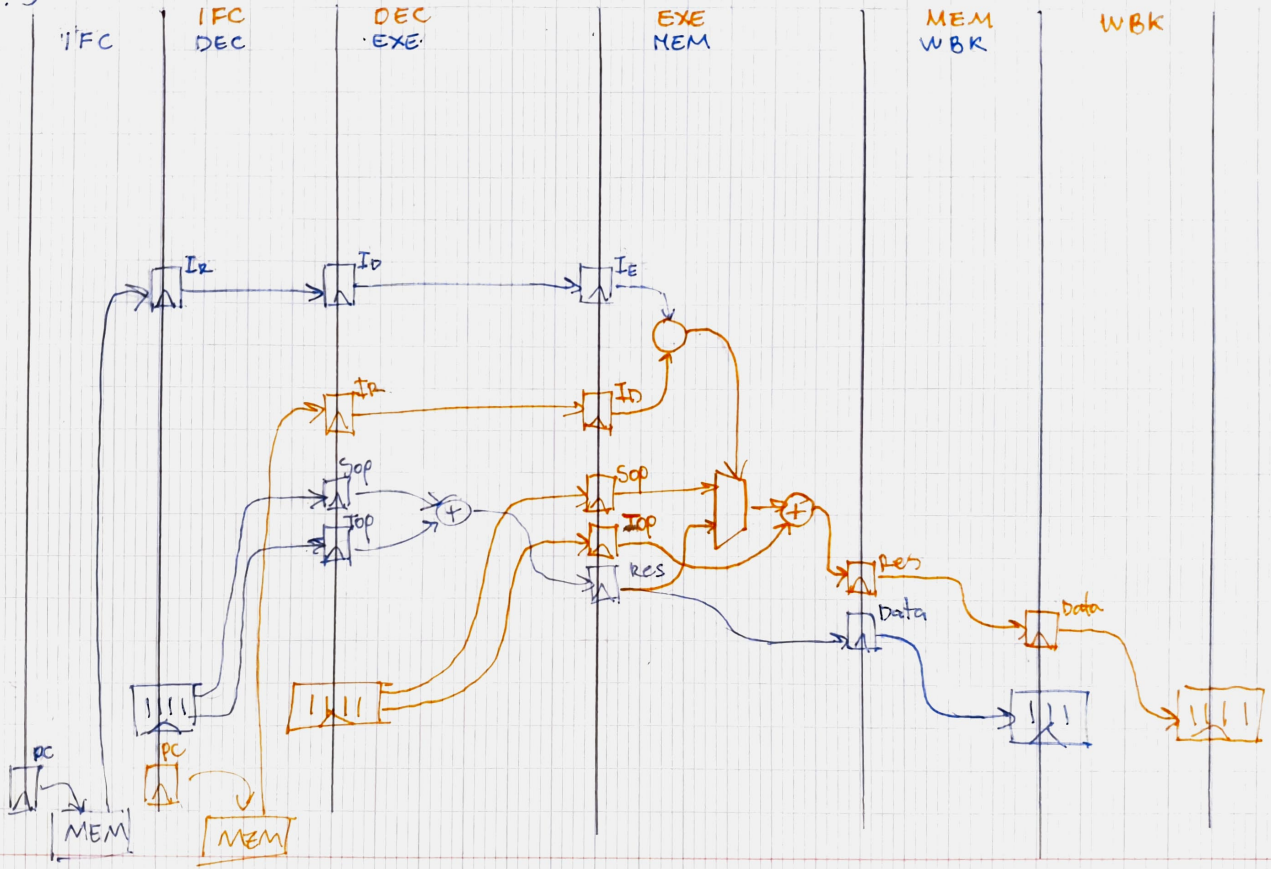


TD3

4/10/23

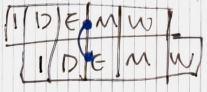
ARCHI TD3

EX. 5



ADD R13, R12, R11
ADD R13, R13, R11
RD RS RT

} On RS



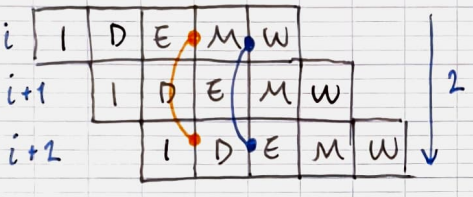
Bypass Orders

Order 1

ADD R13, R12, R11
ADD R14, R13, R15
(R13: RS; R14: RD; R15: RT) } on RS

ADD R13, R12, R11
ADD R14, R13, R13
(R13: RS; R14: RD; R13: RT) } on RT & RS

Order 2



LW R13, 0(R14)
- independent instruction 1
ADD R15, R13, R16
(R13: RT; R15: RD; R16: RS) } on RS

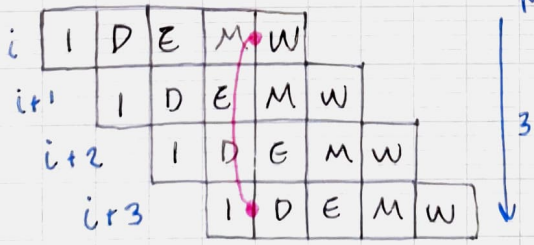
LW R13, 0(R14)
- independent instr. 1
ADD R15, R16, R13
(R13: RT; R15: RD; R16: RS) } on RT

ADD R14, R15, R16
...
BEQ R15, R14, label
(R14: RT; R15: RD) } on RS

ADD R14, R15, R16
...
BEQ R14, R15, label
(R14: RS; R15: RT) } on

Note: R15 is not modified

Order 3

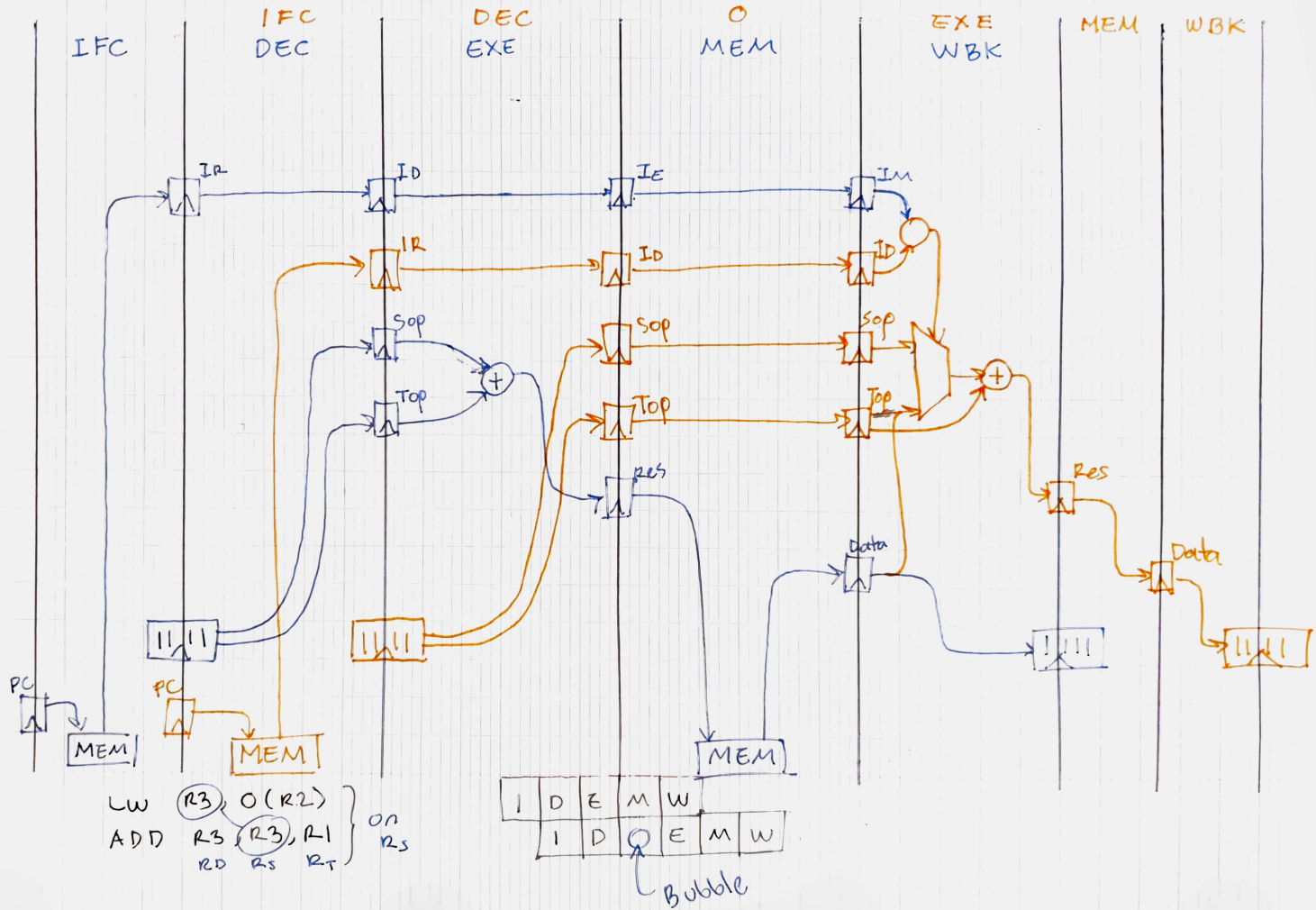


LW R11, 0(R13)
- ind. instr 1
- ind. instr 2
BEQ R12, R11, label
(R12: RS; R11: RT) } on RT

LW R11, 0(R13)
...
...
BEQ R11, R12, label
(R11: RS; R12: RT) } on RS

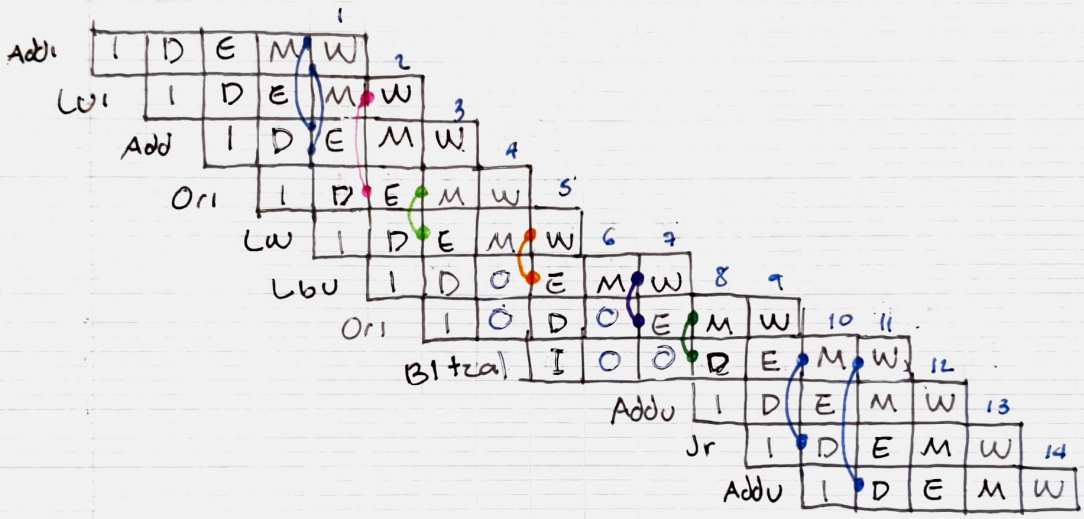
Ex 7

TD3


IFC
DEC
EXE
MEM
WBK
IR
ID
IE
IM
IR
ID
ID
Sop
Sop
Sop
Top
Top
Top
Res
Res
Data
Data
PC
PC
MEM
MEM
MEM
LW R3, 0(R2)
ADD R3, R3, R1
RD Rs RT
ou Rs
I D E M W
I D B E M W
Bubble

Exo 8

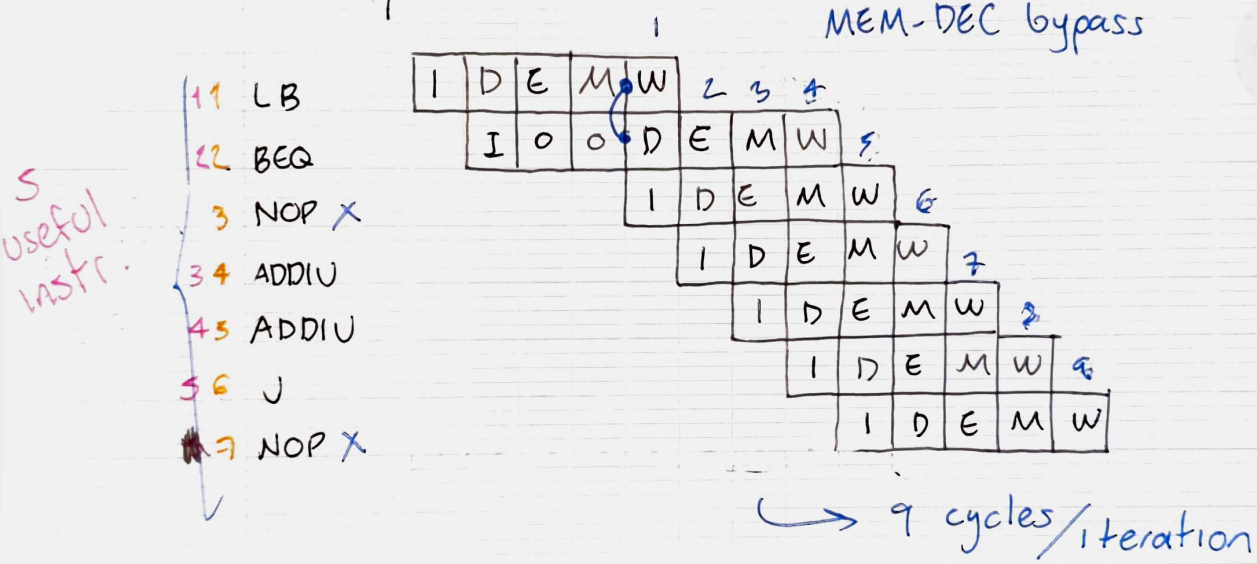
```
1 Addi  r2, r0, 4
2 Lui   r3, 0x000C
3 Add   r2, r2, r2
4 Ori   r3, r3, 0x4568
5 Lw    r2, 0(r3)   ; Calculates @ in EXE cycle
6 Lbu   r2, 0(r2)   ; Looks like order 1 bypass but is order 2
7 Ori   r2, r2, 0x0001 ; Looks like ori 1 but it's order 2
8 Bltzal r2, _next  ; And Link writes the return @ in r31
9 Addu  r0, r0, r0  ; NOP
_next: Jr   r31
       Addu r31, r31, -8
```



Exo 10

1st version

```
-strlen_loop:
    LB    R9, 0(R4)
    BEQ   R9, R0, -end_loop
    NOP
    ADDIU R4, R4, 1
    ADDIU R2, R2, 1
    J     -strlen_loop
    NOP
-end_loop:
```



→ 9 cycles / iteration

9/7 cycles / instruction = CPI

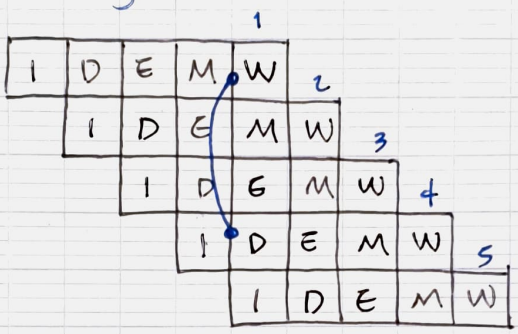
9/5 cycles / useful instruction = CPI useful

2nd version

```
-strlen_loop:
    LB   R9, 0(R4)      (RT: R9, RS: R4)
    ADDIU R4, R4, 1
    ADDI R2, R2, 1
    BNE  R9, R0, -strlen_loop
    NOP                 (RS: R9, RT: R0)
```

Order 3 dependency on RS (MEM-DEC bypass)

1 1 LB
2 2 ADDIU
3 3 ADDI
4 4 BNE
5 NOP



5 cycles / iteration
5/5 cycles / instruction = CPI
5/4 cycles / useful inst. = CPI useful

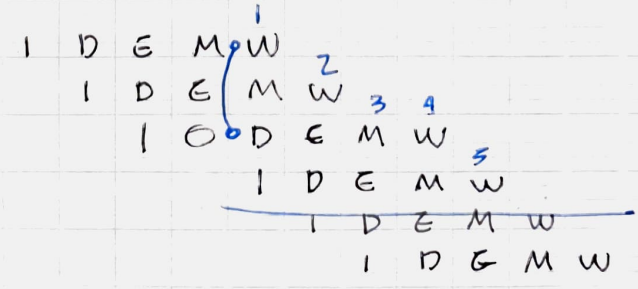
3er version

```
-strlen_loop:
    LB   R9, 0(R4)
    ADDIU R4, R4, 1
    BNE  R9, R0, -strlen_loop
    NOP
    SUBU R2, R8, R4
    JR   R31
```

order 2 dependency on RS

(MEM-DEC bypass)

1 1 LB
2 2 ADDIU
3 3 BNE
4 NOP
SUBU
JR



outside loop

5 cycles / iteration
CPI = 5/4 CPI useful = 5/3

Ex 11

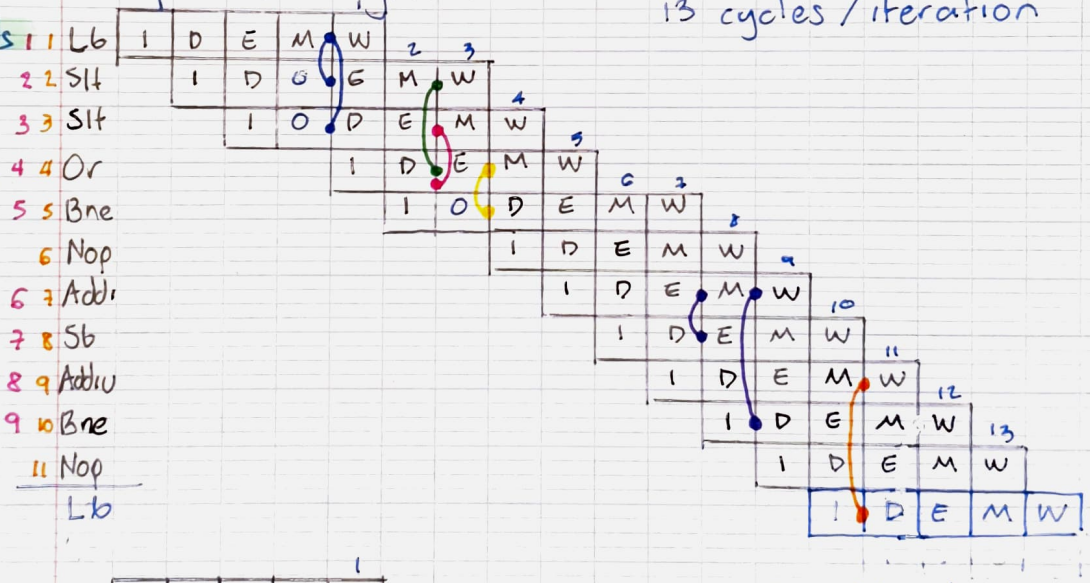
```
        Addu   r2, r0, r4      ; Return value
        Addi   r11, r0, 'a'    ; For comparison
        Addi   r12, r0, 'z'    ; For comparison
-loop:  Lb     r8, 0(r4)       ; read src[i]
        Slt    r9, r8, r11     ; src[i] < 'a'
        Slt    r10, r12, r8    ; 'z' < src[i]
        Or     r10, r10, r9    ; condition
NOP ->  Bne    r10, r0, -endif ; if
        Addi   r8, r8, 'A'-'a' ; Uppercase
        Sb     r8, 0(r4)       ; store src[i]
-endif: Addiu  r4, r4, 1       ; Next address
NOP ->  Bne    r8, r0, -loop
        Jr     r31
        Lb     r2, r0, r4      ; Next iteration
```

Simplified diagram

13 cycles / iteration

If succeeds

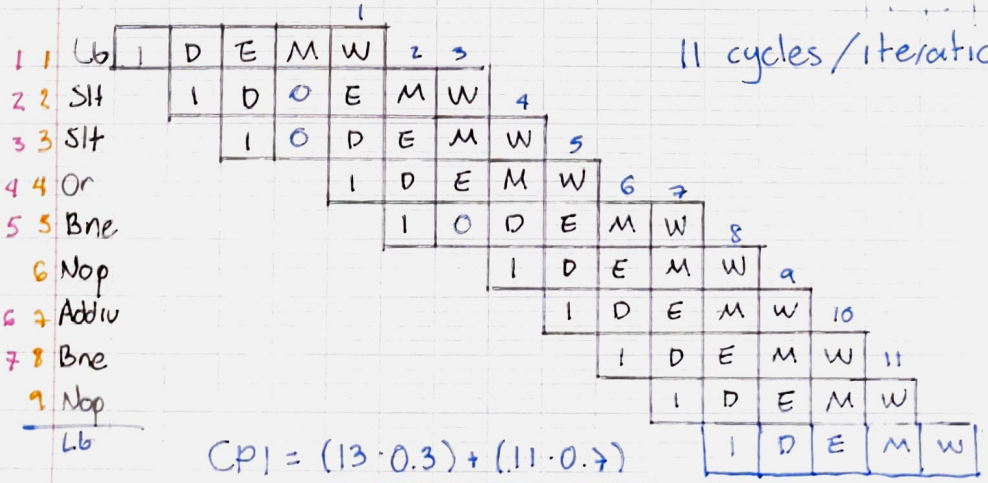
30% of the time



11 cycles / iteration

If fails

100% - 30% → 70% of the time



$$CPI = \frac{(13 \cdot 0.3) + (11 \cdot 0.7)}{(11 \cdot 0.3) + (9 \cdot 0.7)} = 1.20$$

$$CPI_{useful} = \frac{(13 \cdot 0.3) + (11 \cdot 0.7)}{(9 \cdot 0.3) + (7 \cdot 0.7)} = 1.52$$