Architektury systemów komputerowych 2016

Lista zadań nr 12

Na zajęcia 23–25 maja 2016

UWAGA! W trakcie prezentacji zadań należy być przygotowanym do wytłumaczenia haseł, które zostały oznaczone **wytłuszczoną** czcionką.

Na wykładzie pojawił się termin *forwarding*, który powinien być tłumaczony jako „*obejście*". W poniższym tekście nie będzie już występować kalka językowa „~~przekazywanie~~".

W zadaniach odnoszących się do potokowej implementacji procesora `MIPS` zakładamy, że procesor nie implementuje *branch delay slots* chyba, że podano inaczej.

Zadanie 1. Rozważmy jednocyklową implementację procesora `MIPS`, której rysunek został podany na następnej kartce. Przypuśćmy, że jeden z sygnałów kontrolnych:

- `RegWrite`,
- $\texttt{ALUControl}_1$,
- `MemWrite`;

...zostaje błędnie ustawiony w pierwszym wariancie zawsze na `0`, a w drugim na `1`. Które z omówionych na wykładzie instrukcji przestaną działać i dlaczego?

Zadanie 2. Procesory `ARM` należące do rodziny procesorów **RISC** oferują rozkaz dostępu do pamięci z postinkrementacją. Spróbujmy dodać instrukcję `lwinc $rt, imm($rs)` do jednocyklowej implementacji procesora `MIPS`. Jest ona równoważna wykonaniu dwóch rozkazów:

```
lw    $rt, imm($rs)
addi  $rs, $rs, 4
```

Przedstaw propozycję kodowania, stan sygnałów kontrolnych i modyfikacje niezbędne do obsługi tej instrukcji. Jakie problemy sprawia dodanie tego rozkazu?

Zadanie 3. Na potokowej implementacji procesora `MIPS` wykonujemy poniższe ciągi instrukcji:

```
1  lw    $1,40($2)
2  add   $2,$3,$3
3  add   $1,$1,$2
4  sw    $1,20($2)
```

```
1  add   $1,$2,$3
2  sw    $2,0($1)
3  lw    $1,4($2)
4  add   $2,$2,$1
```

- Znajdź wszystkie **zależności danych** typu **Read–After–Write**[1].
- Znajdź wszystkie **hazardy danych**, które wystąpią w implementacji potoku bez i z **obejściami**.
- Narysuj diagram stanu potoku (jak na slajdzie 61) dla wykonania powyższych ciągów instrukcji. Oznacz **wstrzymania potoku** oraz obejścia, z których korzysta przetwarzanie rozkazów.

[1] W procesorach superskalarnych będą jeszcze hazardy danych *Write–After–Read* i *Write–After–Write*.

Zadanie 4. Poniższe ciągi instrukcji wykonujemy na potokowej implementacji procesora MIPS.

```
1  lw    $1,40($6)
2  add   $2,$3,$1
3  add   $1,$6,$4
4  sw    $2,20($4)
5  and   $1,$1,$4
```

```
1  add   $1,$5,$3
2  sw    $1,0($2)
3  lw    $1,4($2)
4  add   $5,$5,$1
5  sw    $1,0($2)
```

- Procesor nie ma zaimplementowanych objeść i **wykrywania hazardów**. Wstaw minimalną ilość rozkazów nop, aby zapewnić poprawność wykonania powyższych ciągów instrukcji.
- Powtórz poprzednie polecenie, ale postaraj się usunąć instrukcje nop. Zachowując semantykę możesz dokonywać dowolnych zmian w kodzie (dodawanie, usuwanie, zmiana kolejności instrukcji). Dodatkowo możesz używać rejestru $7 do przechowywania wartości tymczasowych.
- Procesor implementuje obejścia, ale nie ma wykrywania hazardów. W jaki sposób popsuje się semantyka powyższego kodu?

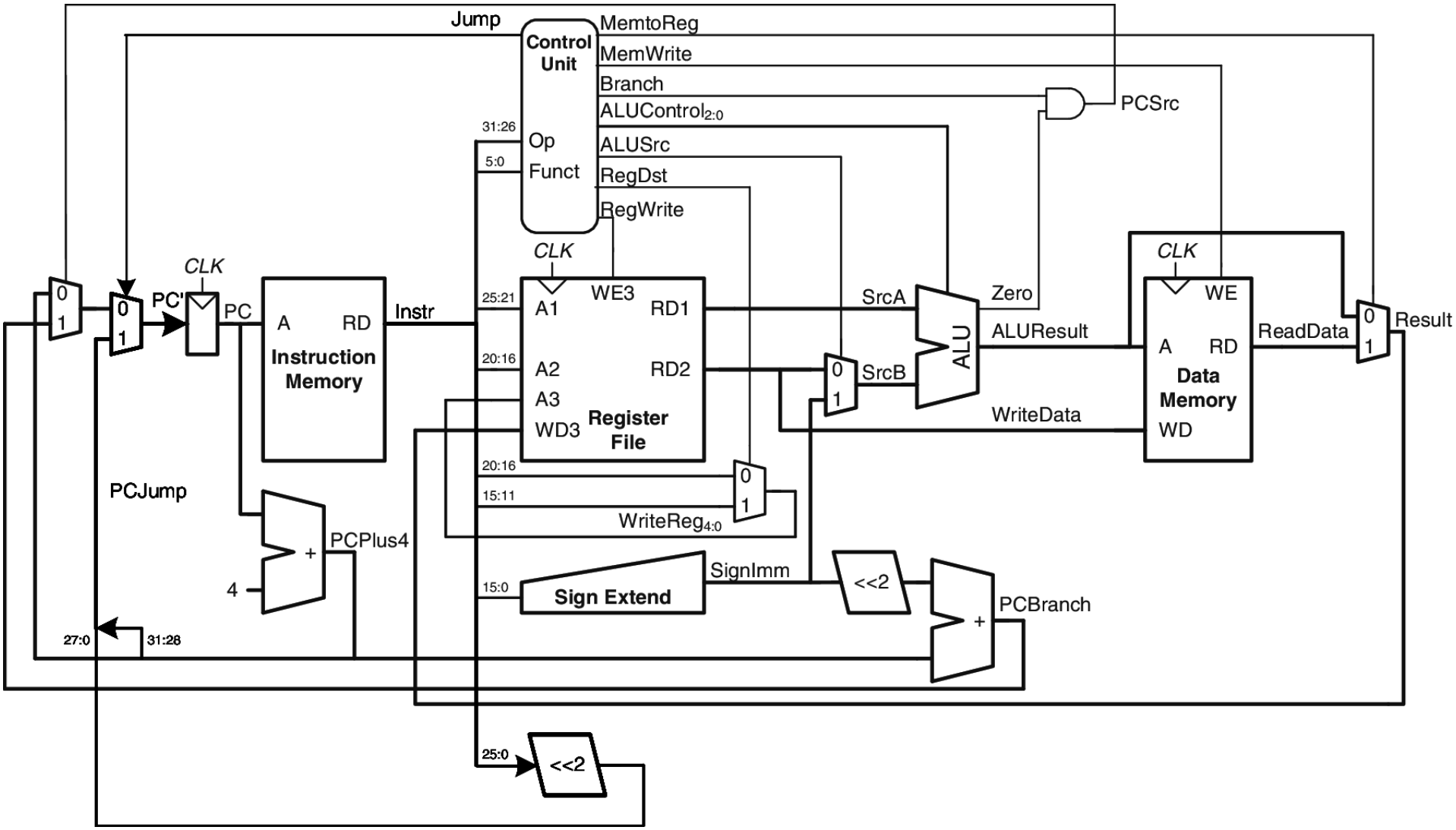
Zadanie 5. Na następnej kartce przedstawiono schemat potokowej implementacji procesora MIPS, do której chcemy dodać obsługę rozkazu slti. Semantykę i kodowanie tej instrukcji podano w dokumencie „*MIPS Reference Data Card*" dostępnym na stronie wykładu. Opisz stan sygnałów kontrolnych i modyfikacje niezbędne do obsłużenia rozkazu slti. Jak należy zmodyfikować jednostkę wykrywania i zapobiegania hazardom?

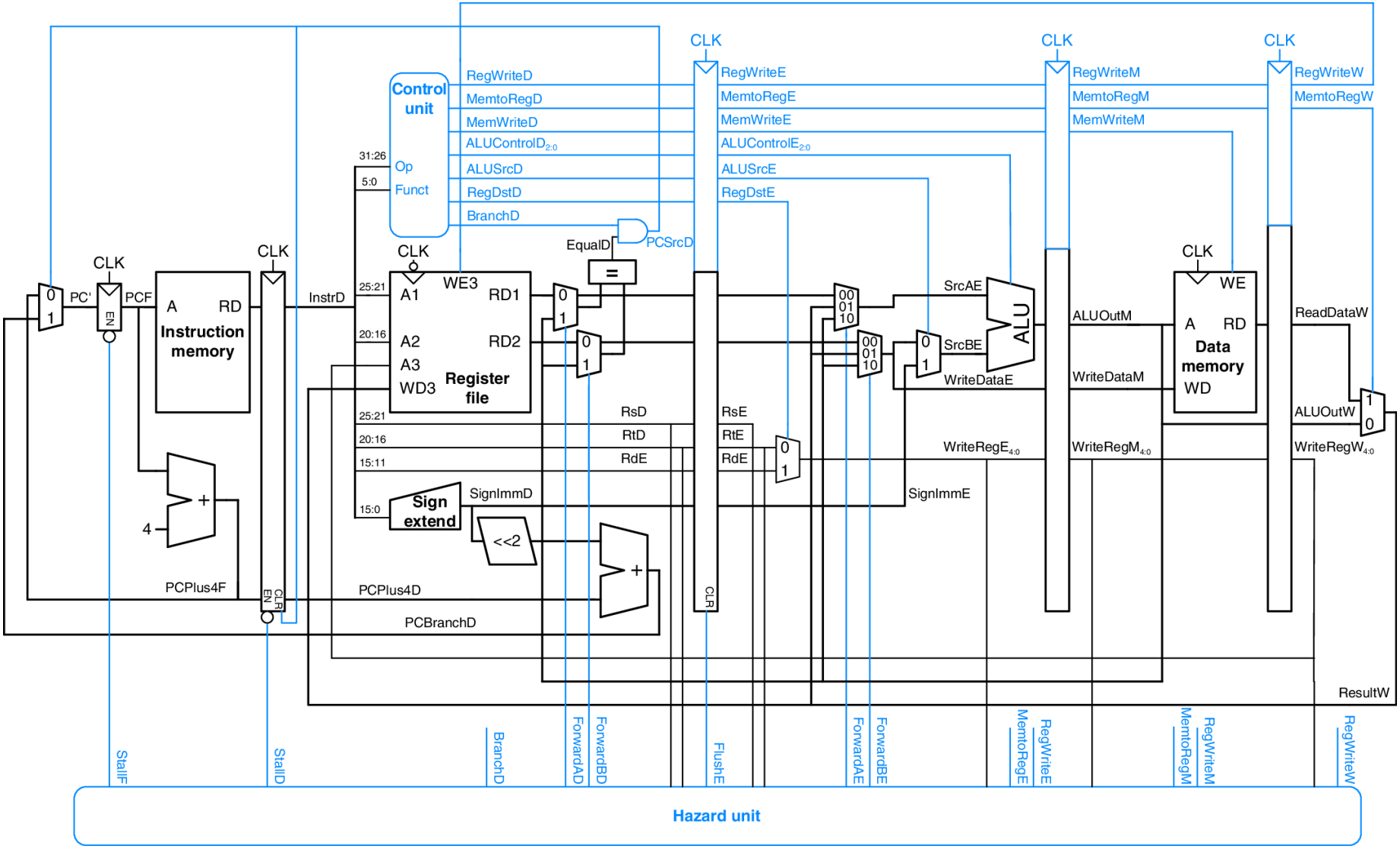
Zadanie 6. Powtórz poprzednie zadanie dla instrukcji skoku j. Zwróć szczególną uwagę na akcję **przeładowania potoku** podejmowaną przez procesor w trakcie wykonywania skoku w fazie EX.

Zadanie 7. Rozważamy procesor potokowy MIPS, którego schemat widnieje na następnej kartce, tj. skoki wykonują się w etapie EX i nie ma ***branch delay slots***.

```
1 l1:
2     addi  $3,$5,4
3     lw    $4,0($6)
4     beq   $2,$3,l2  # (a) skok się wykonał (b) skok się nie wykonał
5     addi  $2,$2,4
6 l2:
7     bne   $2,$4,l1  # (a) skok się nie wykonał (b) skok się wykonał
8     sw    $3,0($2)
9     addi  $3,$2,$3
```

Przyjmujemy strategię **statycznego przewidywania skoków** „*zawsze wykonaj skok*". Dla obydwu wariantów wykonania powyższego ciągu instrukcji narysuj diagram stanu potoku (jak na slajdzie 61).


Jump
Control Unit
MemtoReg
MemWrite
Branch
ALUControl2:0
ALUSrc
RegDst
RegWrite
PCSrc
31:26
5:0
Op
Funct
CLK
CLK
CLK
PC'
PC
A
RD
Instruction Memory
Instr
25:21
20:16
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
SrcA
SrcB
ALU
Zero
ALUResult
WriteData
WE
A
RD
Data Memory
WD
ReadData
Result
PCJump
20:16
15:11
WriteReg4:0
PCPlus4
4
+
15:0
Sign Extend
SignImm
<<2
PCBranch
+
27:0
31:28
25:0
<<2


CLK
Control unit
Op
Funct
31:26
5:0
RegWriteD
MemtoRegD
MemWriteD
ALUControlD2:0
ALUSrcD
RegDstD
BranchD
EqualD
PCSrcD
=
RegWriteE
MemtoRegE
MemWriteE
ALUControlE2:0
ALUSrcE
RegDstE
RegWriteM
MemtoRegM
MemWriteM
RegWriteW
MemtoRegW
PC'
PCF
EN
A
RD
Instruction memory
InstrD
25:21
20:16
A1
A2
A3
WD3
WE3
RD1
RD2
Register file
RsD
RtD
RdE
RsE
RtE
RdE
15:11
15:0
Sign extend
SignImmD
SignImmE
<<2
4
+
PCPlus4F
PCPlus4D
PCBranchD
CLR
SrcAE
SrcBE
WriteDataE
ALU
ALUOutM
WriteDataM
WE
WD
Data memory
ReadDataW
ALUOutW
WriteRegE4:0
WriteRegM4:0
WriteRegW4:0
ResultW
StallF
StallD
BranchD
ForwardAD
ForwardBD
FlushE
ForwardAE
ForwardBE
MemtoRegE
RegWriteE
MemtoRegM
RegWriteM
RegWriteW
Hazard unit