University of California, Berkeley
College of Engineering
Computer Science Division — EECS

Fall 1999 John Kubiatowicz

# Homework Quiz (HW #4)

SOLUTIONS

October 13, 1999

CS152 Computer Architecture and Engineering

This quiz covers one of the problems from homework #4.
Good Luck!

| | |
|---|---|
| Your Name: | |
| SID Number: | |
| Discussion Section: | |

| | |
|---|---|
| Total: | |

# The DoubleSwap instruction

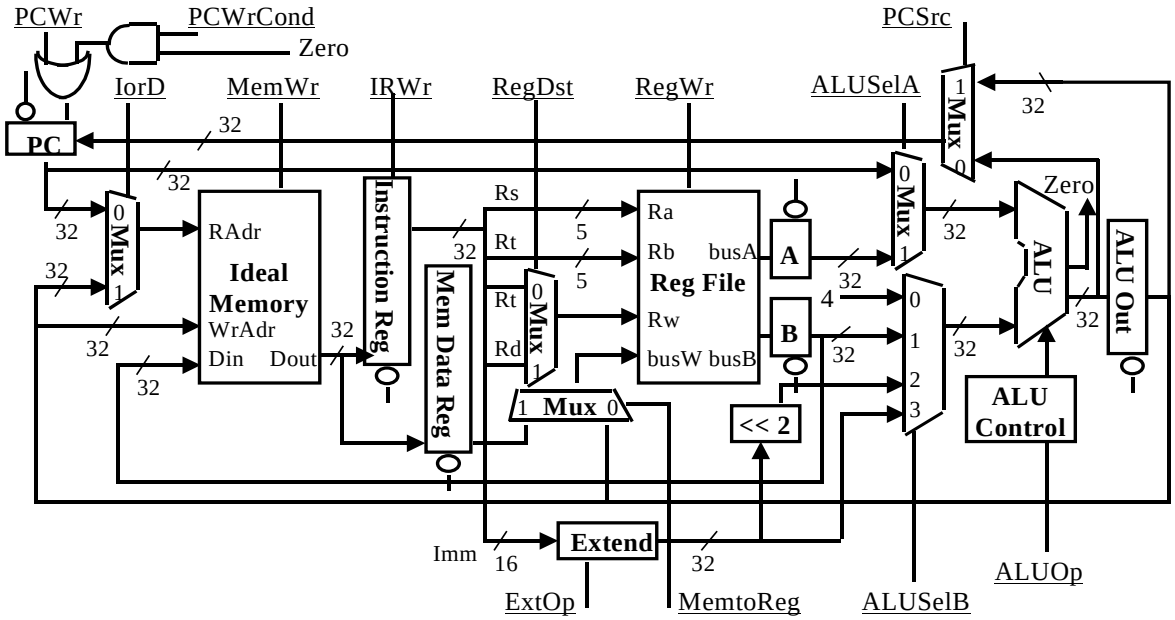



**Figure 1: A multicycle data path**

Figure 1 shows the multicycle datapath from the book. In your homework, you implemented the **`add3`** instruction, which had four registers as operands. You were also asked to implement the **`swap`** instruction which swapped two registers. In this problem, you will implement the `doubleswap` instruction, which takes 4 registers and swaps pairs of values:

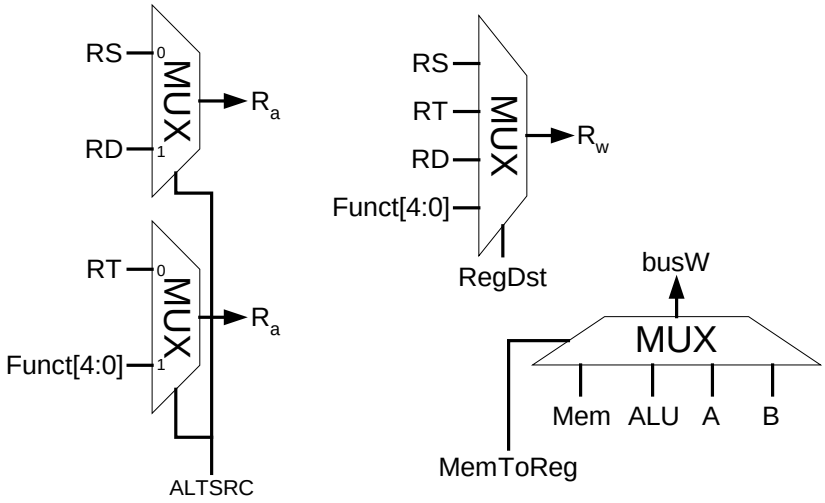
```
doubleswap   $t1,$t2,$t3,$t4
        ⇒ {$t1 ↔ $t2, $t3 ↔ $t4}
```

The coding of this instruction is as follows: $t1 is in the RS field, $t2 is in the RT field, $t3 is in the RD field, and $t4 is in the bottom 5 bits of the instruction (the FUNCT field).

**Problem 1a:** What changes are required to the data path in order to support this instruction? *Note that you must not change the register file block itself; it should have 2 read ports and 1 write port.*

One Solution:

1. Need muxes on the read ports of the register file. In this solution, I have included a single control (ALTSRC) that switches between "normal" and "other".
2. Expand mux on $R_w$ signal (register file destination). We need to be able to select one of 4 different registers to write back.
3. Expand MemToReg mux which selects data to write back to the register file. Need to be able to write back contents of A and B registers.



| Field Name | Values For Field | Function of Field |
|---|---|---|
| ALU | `Add` | ALU Adds |
| | `Sub` | ALU subtracts |
| | `Func` | ALU does function code (Inst[5:0]) |
| | `Or` | ALU does logical OR |
| SRC1 | `PC` | PC ⇒ 1st ALU input |
| | `rs` | R[rs] ⇒1st ALU input |
| SRC2 | `4` | 4 ⇒ 2nd ALU input |
| | `rt` | R[rt] ⇒ 2nd ALU input |
| | `Extend` | sign ext imm16 (Inst[15:0]) ⇒ 2nd ALU input |
| | `Extend0` | zero ext imm16 (Inst[15:0]) ⇒ 2nd ALU input |
| | `ExtShft` | 2nd ALU input = sign extended imm16 << 2 |
| ALU Dest | `rd-ALU` | ALUout ⇒ R[rd] |
| | `rt-ALU` | ALUout ⇒ R[rt] |
| | `rt-Mem` | Mem input ⇒ R[rt] |
| Memory | `Read-PC` | Read Memory using the PC for the address |
| | `Read-ALU` | Read Memory using the ALUout register for the address |
| | `Write-ALU` | Write Memory using the ALUout register for the address |
| MemReg | `IR` | Mem input ⇒ IR |
| PC Write | `ALU` | ALU value ⇒ PCibm |
| | `ALUoutCond` | If ALU Zero is true, then ALUout ⇒ PC |
| Sequence | `Seq` | Go to next sequential microinstruction |
| | `Fetch` | Go to the first microinstruction |
| | `Dispatch` | Dispatch using ROM |

**Problem 1b:** Above, we show the microcode assembly-language and the microcode for 2 of the six instructions that we coded in class *[removed in solutions]*. What changes are needed to the microcode assembly language to support the `doubleswap` instruction?

- *Add new field called RSel with two statements:* ***Norm, ALTSRC***
- *Enhance the ALUDest field to include 4 new statements:*
  ***A-to-rt*** *(A⇒R[rt]),* ***B-to-rs*** *(B⇒R[rs]),* ***A-to-funct*** *(A⇒R[funct[4:0]]),* ***B-to-rd*** *(B⇒R[rd])*

**Problem 1c:** Write complete microcode for `doubleswap`, including the *fetch* and *dispatch* cycles. Be careful not to make changes that would interfere with the other instructions. What is the CPI of your new instruction?

Solution:

| *Label* | *ALU* | *RSel* | *SRC1* | *SRC2* | *ALUDest* | *Memory* | *MemReg* | *PCWrite* | *Sequence* |
|---|---|---|---|---|---|---|---|---|---|
| *Fetch* | *Add* | | *PC* | *4* | | *ReadPC* | *IR* | *ALU* | *Seq* |
| *Dispatch* | *Add* | *Norm* | *PC* | *ExtShft* | | | | | *Dispatch* |
| *dwsap* | | *Norm* | *rs* | *rt* | *A-to-rt* | | | | *Seq* |
| | | *ALTSRC* | *rs* | *rt* | *B-to-rs* | | | | *Seq* |
| | | *ALTSRC* | *rs* | *rt* | *A-to-funct* | | | | *Seq* |
| | | | *rs* | *rt* | *B-to-rd* | | | | *Fetch* |

*CPI for new instruction = 6. Note that by asserting "Norm" in the dispatch cycle, we make sure that all of the other instructions will have the right values in their "A" and "B" registers during the first cycle after dispatch (which is where they all use A and B).*