Project Report

Reem Alansary, Menna Ibrahim, Leqaa Jamal, Lydia Youssef, Maya Eraky

May 16, 2020

CSEN601 Computer Architecture: Architecture Design Project

1 Our Code Features

1. **Microarchitecture:**
   - We chose von Neumann architecture.
2. **Instruction Memory and Data Memory Size:**
   - One memory for data and instructions
   - 1024 x 32-bit
3. **Total Number of Registers:**
   - *16 registers*
     - a zero register which is always 0
     - pc register
     - 14 temporary registers
4. **Instruction Set:**
   - *Instruction set 2*
     - Arithmetic Instructions: 1. Sub. 2. Add. 3. Add immediate. 4. Multiply.
     - Logical Instructions: 1. Or. 2. And immediate. 3. Shift right logical. 4. Shift left logical.
     - Data Transfer Instructions: 1. Load word. 2. Store word.
     - Conditional Branch Instructions: 1. Branch on equal. 2. Branch on less than.
     - Comparison Instructions: 1. Set on less than immediate.
     - Unconditional Jump Instructions: 1. Jump Register.
5. **Type of Cache and Replacement Policy Used:**
   - direct mapped cache
6. **Signals:**
   - *memread;memwrite;regwrite;pcsrc;regdst;memtoreg;Alusrc;branch;jump;*
7. **Stages Caches:**
   - cache1;cache2;cache3;cache4
   - We used ‘cache 1’ to pass information from stage fetch to decode, ‘cache 2’ to pass information from stage decode to execution ,’cache 3’ to pass information from stage execution to memory stage and ‘cache4’ to pass information from memory stage to write back

2 Datapath of our Architecture

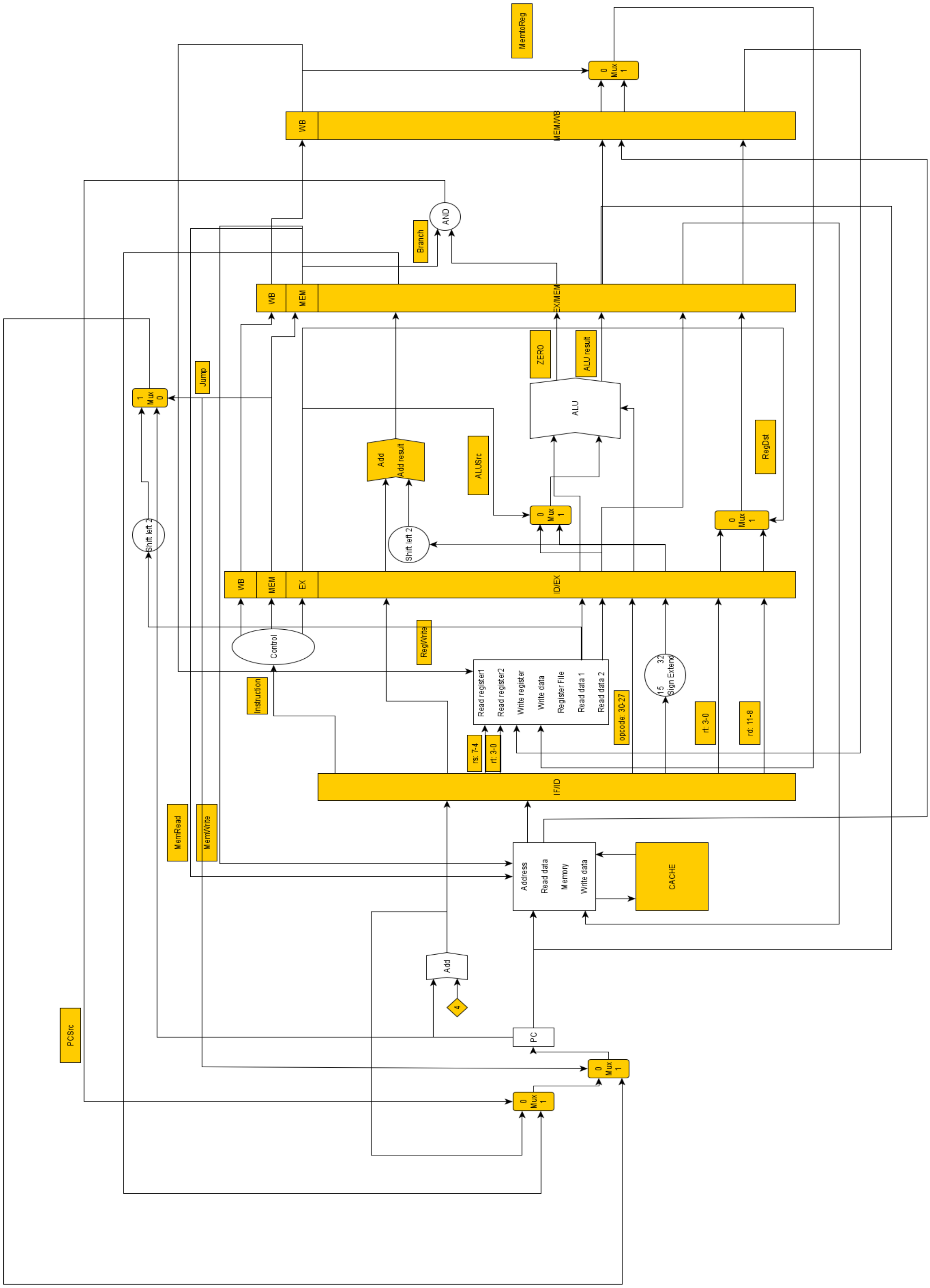

Figure 1: Datapath

3 Code Structure and Flow

1. **Structure**

   Our code is divided into 6 classes:

   Main contains the required variables to save values that will be used by multiple class and has the main logic to run the instructions in a pipelined manner

   FetchInstruction fetches the next instruction and increments the pc

   DecodeInstruction decodes the current instruction and assigns the decoded values to their corresponding variables in the main class to be used by other classes

   ExecuteInstruction executes the decoded instruction

   Memory provides access to both memory and cache when an instruction should read or write to memory

   WriteBackInstruction writes a value back to the current chosen destination register

2. **Flow**

   At the very beginning the program counter for the small program that will be entered is initialized. Afterwards, our program reads in the instructions, entered in binary form and on separate lines that are terminated with -1 on a line, and stores them in memory. Then, the main loop begins execution; the main loop is the only loop in the main class and it is a while loop. To effectively achieve the pipelined manner of execution, the main loop calls the 5 stages in reverse order; this means that writeback is executed first and fetch is executed last. However, the output of each stage is saved in a string variable in the main class so that after every cycle the output is ordered. In the FetchInstruction stage, the instruction at the index specified by the current value of the program counter is fetched from memory and passed over to the next stage. After fetching and before moving on to the decode stage the value of the program counter is incremented. In the DecodeInstruction stage, the fetched instruction is decoded according to our instruction format and all control signals are generated for the current instruction. Lastly, this stage passes on all the information needed by the next stage. The next stage, which is ExecuteInstruction, receives information from the decode stage and proceeds with execution according to the decoded opcode of the instruction just received. In the case of a branch or a jump instruction, the execute stage also adds the immediate value to the program counter. Both DecodeInstruction and ExecuteInstruction stages pass on the control signals required by coming stages to them. At the Memory stage, if the read signal is on, then we must fill up the cache with the needed data. In case the data at the desired index is missing, then the cache is filled up with the ALUresult, otherwise, the data is fetched from the cache to be read. In case the index does have data, but not the needed one, then the new data will replace the old one. If the write signal is on, then the data given will override anything in the cache in its index, and will be saved in the memory in write data. If both signals are off then the ALUResult will just get saved in the cache. The cache is then passed to the WriteBackInstruction stage where the memtoreg value decides between read data and write data to write back.

4 Screenshots of Some Test Cases

```
00011000000000000101100000000000 //addim reg8 = 5 immediate +value in reg0
01000000000000001001100110000000 //lw reg9 = memo[ value in reg8 + immediate 9]
// the index which is [immediate+rs] for memo in lw should be more than the num of instructions you entered.
00000000000000000000110010001001 //add reg12 = value in reg8 + value in reg9
00100000000000000000011111011110 //or reg7 = value in reg13 + value in reg14
01001000000000001000000010001001 //sw memo[value in reg8 + immediate 8] = value in reg9
// the index which is [immediate+rs] for memo in sw should be more than the num of instructions you entered.
01010000000000000001000001100100 //beq if(value in reg12 == value in reg12) nextPC = nextPC+ immediate 2.
```

```
Console
&lt;terminated&gt; Main (2) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_131.jdk/Contents/Home/bin/java (May 15, 2020, 10:42:23 PM)
Please enter each instruction as a binary number and terminate instruction list
with -1, written as a decimal number.
00011000000000000101100000000000
01000000000000001001100110000000
00000000000000000000110010001001
00100000000000000000011111011110
01001000000000001000000010001001
01010000000000000001000001100100
-1
```

Figure 2: Test Cases

```
IN CYCLE: 1
00011000000000000101100000000000 in Fetch Stage.
Next PC: 00000000000000000000000000000001
Instruction: 00011000000000000101100000000000

IN CYCLE: 2
01000000000000001001100110000000 in Fetch Stage.
Next PC: 00000000000000000000000000000010
Instruction: 01000000000000001001100110000000

00011000000000000101100000000000 in Decode Stage.
read data 1: 00000000000000000000000000000000
read data 2: 00000000000000000000000000000000
sign-extend: 10000000000000000000000000000101
Next PC: 00000000000000000000000000000001
rt: 0000
rd: 1000
WB Controls: MemtoReg: 0 RegWrite: 1
MEM Controls: MemRead: 0 MemWrite: 0 Branch: 0 Jump: 0
EX Controls: RegDst: 1 ALUOp: 0011 ALUSrc: 1
PCsrc: 0
```

Figure 3: Output 1

```
IN CYCLE: 3
00000000000000000000110010001001 in Fetch Stage.
Next PC: 00000000000000000000000000000011
Instruction: 00000000000000000000110010001001

01000000000000001001100110000000 in Decode Stage.
read data 1: 00000000000000000000000000001000
read data 2: 00000000000000000000000000000000
sign-extend: 10000000000000000000000000001001
Next PC: 00000000000000000000000000000010
rt: 0000
rd: 1001
WB Controls: MemtoReg: 1 RegWrite: 1
MEM Controls: MemRead: 1 MemWrite: 0 Branch: 0 Jump: 0
EX Controls: RegDst: 1 ALUOp: 1000 ALUSrc: 1
PCsrc: 0

00011000000000000101100000000000 in Execute Stage.
zero flag: 0 branch address: 00000000000000000000000000000001
ALU result/address: 101
register value to write to memory:  0000 0000 0000 0000 0000 0000 0000 0000
rt/rd register: 8
WB controls: MemToReg: 0 RegWrite:1
MEM controls: MemRead: 0, MemWrite:  0 ,Branch: 0 ,pcsrc:0
```

Figure 4: Output 2

```
IN CYCLE: 4
00100000000000000000011111011110 in Fetch Stage.
Next PC: 00000000000000000000000000000100
Instruction: 00100000000000000000011111011110

00000000000000000000110010001001 in Decode Stage.
read data 1: 00000000000000000000000000001000
read data 2: 00000000000000000000000000001001
sign-extend: 10000000000000000000000000000000
Next PC: 00000000000000000000000000000011
rt: 1001
rd: 1100
WB Controls: MemtoReg: 0 RegWrite: 1
MEM Controls: MemRead: 0 MemWrite: 0 Branch: 0 Jump: 0
EX Controls: RegDst: 1 ALUOp: 0000 ALUSrc: 0
PCsrc: 0

01000000000000001001100110000000 in Execute Stage.
zero flag: 0 branch address: 00000000000000000000000000000010
ALU result/address: 10001
register value to write to memory:  0000 0000 0000 0000 0000 0000 0000 0000
rt/rd register: 9
WB controls: MemToReg: 1 RegWrite:1
MEM controls: MemRead: 1, MemWrite:  0 ,Branch: 0 ,pcsrc:0

00011000000000000101100000000000 in Memory Stage.
mem variables
ALU result: 5
memory word read: don't care
rt/rd field: 8 WB controls: MemToReg: 0, RegWrite: 1
```

Figure 5: Output 4

```
IN CYCLE: 5
01001000000000001000000010001001 in Fetch Stage.
Next PC: 00000000000000000000000000000101
Instruction: 01001000000000001000000010001001

00100000000000000000011111011110 in Decode Stage.
read data 1: 00000000000000000000000000001101
read data 2: 00000000000000000000000000001110
sign-extend: 10000000000000000000000000000000
Next PC: 00000000000000000000000000000100
rt: 1110
rd: 0111
WB Controls: MemtoReg: 0 RegWrite: 1
MEM Controls: MemRead: 0 MemWrite: 0 Branch: 0 Jump: 0
EX Controls: RegDst: 1 ALUOp: 0100 ALUSrc: 0
PCsrc: 0

00000000000000000000110010001001 in Execute Stage.
zero flag: 0 branch address: 00000000000000000000000000000011
ALU result/address: 10001
register value to write to memory:  0000 0000 0000 0000 0000 0000 0000 0000
rt/rd register: 12
WB controls: MemToReg: 0 RegWrite:1
MEM controls: MemRead: 0, MemWrite:  0 ,Branch: 0 ,pcsrc:0

01000000000000001001100110000000 in Memory Stage.
mem variables
ALU result: 17
memory word read: 0
rt/rd field: 9 WB controls: MemToReg: 1,  RegWrite: 1

00011000000000000101100000000000 in WB Stage
```

Figure 6: Output 5

```
IN CYCLE: 6
01010000000000000010000011001100 in Fetch Stage.
Next PC: 00000000000000000000000000000110
Instruction: 01010000000000000010000011001100

01001000000000001000000010001001 in Decode Stage.
read data 1: 00000000000000000000000000000101
read data 2: 00000000000000000000000000000000
sign-extend: 10000000000000000000000000001000
Next PC: 00000000000000000000000000000101
rt: 1001
rd: 0000
WB Controls: MemtoReg: 0 RegWrite: 0
MEM Controls: MemRead: 0 MemWrite: 1 Branch: 0 Jump: 0
EX Controls: RegDst: 0 ALUOp: 1001 ALUSrc: 1
PCsrc: 0

00100000000000000000011111011110 in Execute Stage.
zero flag: 0 branch address: 00000000000000000000000000000100
ALU result/address: 1111
register value to write to memory:  0000 0000 0000 0000 0000 0000 0000 0000
rt/rd register: 7
WB controls: MemToReg: 0 RegWrite:1
MEM controls: MemRead: 0, MemWrite:  0 ,Branch: 0 ,pcsrc:0

00000000000000000000110010001001 in Memory Stage.
mem variables
ALU result: 17
memory word read: don't care
rt/rd field: 12 WB controls: MemToReg: 0, RegWrite: 1

01000000000000001001100110000000 in WB Stage
```

Figure 7: Output 6

```
IN CYCLE: 7
01010000000000000010000011001100 in Decode Stage.
read data 1: 00000000000000000000000000010001
read data 2: 00000000000000000000000000010001
sign-extend: 10000000000000000000000000000010
Next PC: 00000000000000000000000000000110
rt: 1100
rd: 0000
WB Controls: MemtoReg: 0 RegWrite: 0
MEM Controls: MemRead: 0 MemWrite: 0 Branch: 1 Jump: 0
EX Controls: RegDst: 0 ALUOp: 1010 ALUSrc: 0
PCsrc: 0

01001000000000001000000010001001 in Execute Stage.
zero flag: 0 branch address: 00000000000000000000000000000101
ALU result/address: 1101
register value to write to memory:  0
rt/rd register: 9
WB controls: MemToReg: 0 RegWrite:0
MEM controls: MemRead: 0, MemWrite:  1 ,Branch: 0 ,pcsrc:0

00100000000000000000011111011110 in Memory Stage.
mem variables
ALU result: 15
memory word read: don't care
rt/rd field: 7 WB controls: MemToReg: 0, RegWrite: 1

00000000000000000000110010001001 in WB Stage
```

Figure 8: Output 7

```
IN CYCLE: 8
01010000000000000010000011001100 in Execute Stage.
zero flag: 1 branch address: 00000000000000000000000000001000
ALU result/address: 0
register value to write to memory:  0000 0000 0000 0000 0000 0000 0000 0000
rt/rd register: 12
WB controls: MemToReg: 0 RegWrite:0
MEM controls: MemRead: 0, MemWrite:  0 ,Branch: 1 ,pcsrc:1

01001000000000001000000010001001 in Memory Stage.
mem variables
ALU result: 13
memory word read: don't care
rt/rd field: 9 WB controls: MemToReg: 0, RegWrite: 0

00100000000000000000011111011110 in WB Stage

IN CYCLE: 9
01010000000000000010000011001100 in Memory Stage.
mem variables
ALU result: 0
memory word read: don't care
rt/rd field: 12 WB controls: MemToReg: 0, RegWrite: 0

01001000000000001000000010001001 in WB Stage

IN CYCLE: 10
01010000000000000010000011001100 in WB Stage
```

Figure 9: Output 8

5 Points and Notes

- The order of speakers in the video is 1) Menna Ibrahim, 2) Lydia Youssef, 3) Maya Eraky, 4) Leqaa Jamal, 5) Reem Alansary.
- We have sent the video inside a pptx file as requested on piazza, therefore you MUST download it from the drive for it to work properly.
- We have updated the data-path and the java implementation so use the new ones.
- When testing the code If you want to make a SW or LW instructions please make sure that the index (which the value will be stored in or loaded from) is greater than the number of instructions you added, as we are using 'Von Neumann' and the place where data are stored is after the instructions in the same memory.
- The branch address that appears in the output is the address of the PC+1+branch so if the condition of the branch is false that value is not important and shall be ignored.
- The PCsrc signal will be equal to 1 if and only if the condition of the branch is true, but the branch signal will be always 1 if the instruction is actually a branch instruction and zero otherwise.
- In execute if there is a data to be written on the memory that data will appear in the output as integer, if there is no data to be written on the memory this section will appear as 32 bits of 0s.
- When you finish with putting the instructions in the input you should put -1 to terminate giving the input and start to have the output.
- Every register will contain its number at the start of the program, for example `registers[1]=1`.

6 Appendix

Register	Usage
$0	contains constant value of zero
$1-$14	temporaries
$15	PC

Arithmetic instruction	1 bit	4 bits	15 bits	4 bits	4 bits	4 bits
Sub	0	Opcode (0001)	Immediate (Don't care)	Dest	rss	rtt
Add	0	Opcode (0000)	Immediate (Don't care)	Dest	rss	rtt
Add immediate	0	Opcode (0011)	Immediate (Don't care)	Dest	rss	rtt (Don't care)
Multiply	0	Opcode (0010)	Immediate (Don't care)	Dest	rss	rtt

Logical instruction	1 bit	4 bits	15 bits	4 bits	4 bits	4 bits
Or	0	Opcode (0100)	Immediate (Don't care)	Dest	rss	rtt
And immediate	0	Opcode (0101)	Immediate	Dest	rss	rtt (Don't care)
Shift right logical	0	Opcode (0110)	Immediate	Dest	rss	rtt (Don't care)
Shift left logical	0	Opcode (0111)	Immediate	Dest	rss	rtt (Don't care)

Data Transfer instruction	1 bit	4 bits	15 bits	4 bits	4 bits	4 bits
Lw	0	Opcode (1000)	Immediate	Dest	rss	rtt (Don't care)
sw	0	Opcode (1001)	Immediate	Dest (Don't care)	rss	rtt

Conditional branch instruction	1 bit	4 bits	15 bits	4 bits	4 bits	4 bits
Branch on equal	0	Opcode (1010)	Immediate	Dest (Don't care)	rss	rtt
Branch on less than	0	Opcode (1011)	Immediate	Dest (Don't care)	rss	rtt

Comparison instruction	1 bit	4 bits	15 bits	4 bits	4 bits	4 bits
Set on less than immediate	0	Opcode (1100)	Immediate	Dest	rss	rtt (Don't care)

Unconditional jump instruction	1 bit	4 bits	15 bits	4 bits	4 bits	4 bits
Jump Register	0	Opcode (1101)	Immediate (Don't care)	Dest (Don't care)	rss	rtt (Don't care)