Computer System Architecture

CSEN601

Supervisor: Assoc. Prof. Dr. Hassan Soubra

Department: Media Engineering & Technology

Project Proposal Milestone 1

Team Name: Newbies

Team Members:

Islam Nasr 40-4936

Kariman Hossam 40-1398

Hana Kamal 40-1183

Gasser Khaled 40-18970

Ahmed Salah 40-1316

1- RISC or CISC?

In our implementation of our version of the ISA we decided to implement it in RISC Architecture because it is simpler in implementation than CISC. RISC also helps easier programming of our basic instructions, in addition to that RISC emphasizes cycles per instruction. RISC instructions focuses on software rather than hardware, which helps in making a software which is more efficient with fewer instructions than that of CISC.

2- Von Neumann or Harvard?

We decided in our implementation to use Von Neumann over Harvard because in Von Neumann we have same memory for instructions and data but different location where instruction are stored in specific locations and the data are in the others. We decided to use Von Neumann after we found from the research that Von Neumann is more efficient as we are using most of the memory we saved for our ISA and we are not wasting a large space.

Register Details:

Register Number	Register Name	Holding
0	$zero	Holds the value 0
1	$one	Holds the value 1
2	$sp	Holds the location of current stack
3	$ra	Holds the return address
4-7	$s0→$s3	Values not overwriting usually saved here
8-11	$t0→$t1	Values used usually and overwritten
12-14	$a0 → $a2	Arguments saved for procedures to use
15	$v0	Output register

3- Memory size:

We decided to use memory size of 2^{16} because we use 2^{12} bits for the code and the rest are for the instructions.

4- Instruction format:

R type register:

Opcode (4 bits)	RD (4 bits)	RS (4 bits)	RT (4 bits)

I type register:

Opcode (4 bits)	RD (4 bits)	Immediate (8 bits)

J type register:

Opcode (4bits)	Offset (12 bits)

Instructions:

Read Immediate:

Type: I

Format: RDI rd, immediate

Opcode: 0000

Purpose: Input a word into a register

Description: An instruction that takes as input an immediate of 8 bits and stores it in a register. Rd is that destination register and the immediate is am 8-bit signed integer.

Read:

Type: R

Format: RD rd, offset(rs)

Opcode: 0001

Purpose: Read a word from memory.

Description: An instruction that uses register rd as the destination register to which it will write the value that is stored at the memory location stored in register rs (which is in the data region of the memory). The offset is just an index that could be added to the value of rs when fetching the word in rs. The offset gets multiplied by 2 because our memory of choice is byte addressable.

Write:

Type: R

Format: WR rd, rs, rt

Opcode: 0010

Purpose: Write a word to memory.

Description: An instruction that uses the value inside register rd as the memory base address to which it will store the value inside the register rs. Register rt is used as the offset, to which it could be added to rd which contains the memory base address I want to store in.

Sum:

Type: R

Format: SUM rd, rs, rt

Opcode: 0011

Purpose: To perform the addition operation.

Description: Takes the signed value inside register rs and rt, adds them and stores the result inside register rd. It does this by sign extending the value inside rs and rt to write a 16 bit word into rd.

Difference:

Type: R

Format: DIFF rd, rs, rt

Opcode: 0100

Purpose: To perform the subtraction operation.

Description: Subtracts the signed value inside of register rt from the signed value inside of register rs and writes the result inside register rd. It does this by sign extending the values of rs and rt to write a 16-bit word into rd.

AND:

Type: R

Format: AND rd, rs, rt

Opcode: 0101

Purpose: Performs bitwise AND operation.

Description: Takes the binary representation of the values inside the register rs and rt, performs the bitwise AND operation then stores the result into register rd.

OR:

Type: R

Format: OR rd, rs, rt

Opcode: 0110

Purpose: Performs the bitwise OR operation.

Description: Takes the binary representation of the values inside the register rs and rt, performs the bitwise OR operation then stores the result into register rd.

NOT:

Type: R

Format: NOT rd, rs

Opcode: 0111

Purpose: Performs the bitwise NOT operation.

Description: Takes the binary representation of the value inside the register rs, performs the bitwise NOT operation then stores the result into register rd.

HOP

Type: J

Format: HOP address

Opcode: 1000

Purpose: To branch within the current 8KB region.

Description: This is a PC-region branch. The effective destination address is in the current 8KB region. The low 12 bits are for the destination address and the upper four bits are for the opcode for this instruction. Hop to the destination address and execute the instruction that follows the hop.

HOP REGISTER

Type: R

Format: HOPR rs

Opcode: 1001

Purpose: To execute a branch to an instruction address in a register.

Description: Hop to the effective address in rs and execute the instruction that follow the hop.

HOP AND TIE

Type: J

Format: HAT address

Opcode: 1010

Purpose: To execute a procedure call within the 8KB region of instructions.

Description: Place the return address in the GPR ra. The return tie is the address of the next instruction following the branch. This is a PC-region branch. The effective destination address is in the current 8KB region. The low 12 bits are for the destination address and the upper four bits are for the opcode for this instruction. Hop to the destination address and execute the instruction that follows the hop.

SET MINIMUM

Type: R

Format: SMN rd, rs, rt

Opcode: 1011

Purpose: To determine the minimum of two registers.

Description: Sets the destination register to the minimum of of GPRs rs and rt by comparing their contents.

SET MAXIMUM

Type: R

Format: SMX rd, rs, rt

Opcode: 1100

Purpose: To determine the maximum of two registers.

Description: Sets the destination register to the maximum of GPRs rs and rt by comparing their contents.

SET LESS THAN

Type: R

Format: SLT rd, rs, rt

Opcode: 1100

Purpose: To record the result of a less-than comparison.

Description: Compares the contents of GPRs rs *and* rt as signed integers and records the boolean result of the comparison in GPR rd. If GPR rs is less than GPR *rt*, the result is 1; otherwise, it is 0.

SHIFT LEFT LOGICAL

Type: I

Format: SLL rd, immediate

Opcode: 1110

Purpose: To shift left a word by a specific shift amount.

Description: The contents of the low-order 16-bit word of GPR rd are shifted left by inserting zeros into the emptied bits. The result is placed in GPR rd. The bit-shift amount is specified by the immediate value.

SKIP IF EQUAL

Type: R

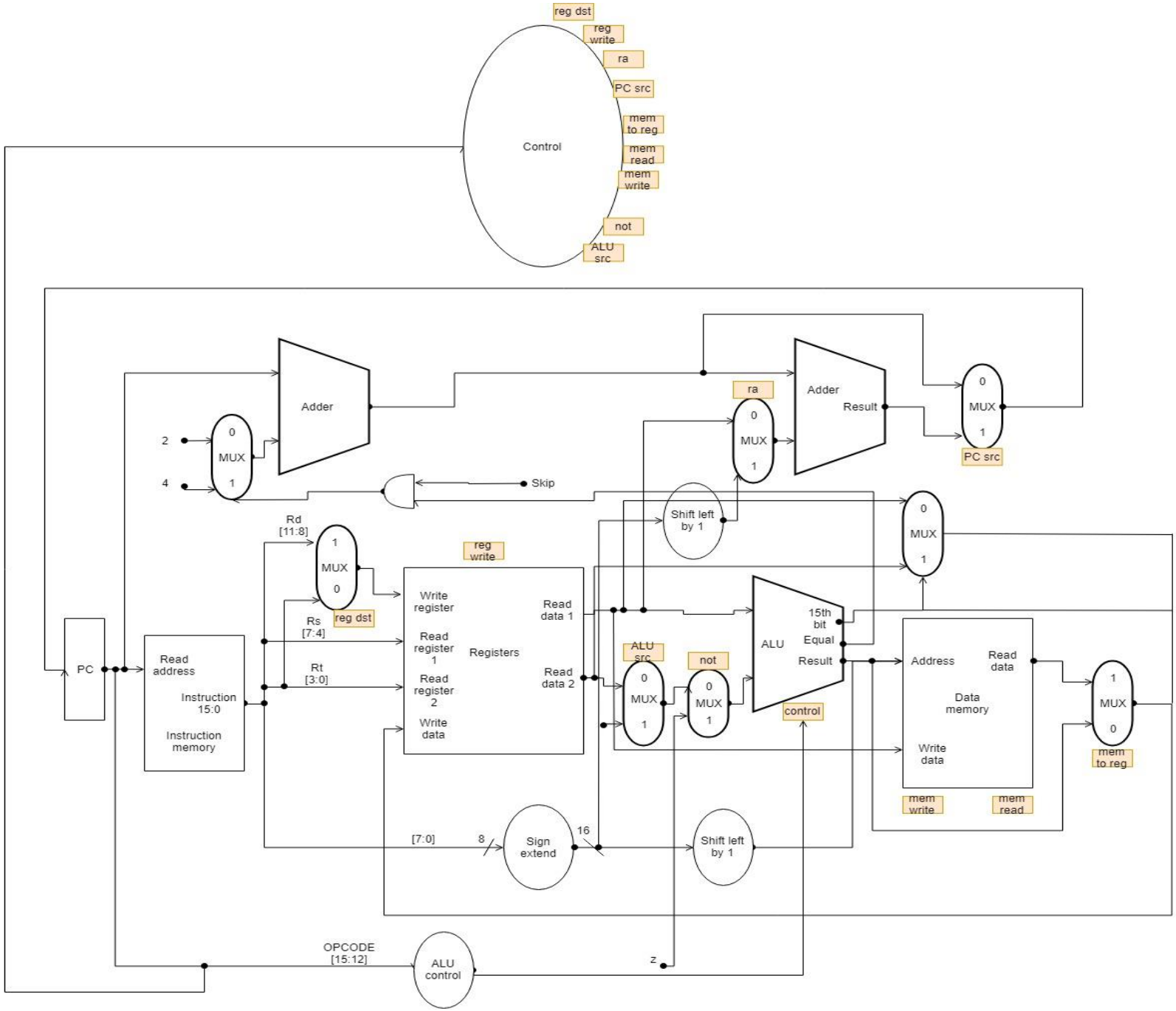
Format: SKP rs,rt

Opcode: 1111

Purpose: To skip on equal by comparing GPRs rs and rt then do a PC-relative conditional branch.

Description: If the contents of GPRs rs and rt are equal, increment the program counter by two more to skip the next instruction to be fetched.

Data path:


reg dst
reg write
ra
PC src
mem to reg
mem read
mem write
not
ALU src
Control
Adder
2
4
MUX
Skip
Adder
Result
ra
PC src
Shift left by 1
Rd [11:8]
reg write
reg dst
Rs [7:4]
Rt [3:0]
Write register
Read register 1
Read register 2
Write data
Registers
Read data 1
Read data 2
PC
Read address
Instruction 15:0
Instruction memory
ALU src
not
ALU
15th bit
Equal
Result
control
Address
Read data
Data memory
Write data
mem write
mem read
mem to reg
[7:0]
8
Sign extend
16
Shift left by 1
OPCODE [15:12]
ALU control
z