CA PROJECT

Team Name: Overloadz

Team Members:

Samar Shokry 43-1746

Ahmed Ossama 43-3823

Omar Ads 43-12699

Nourhan Salah 43-5139

Aly Elshamy 43-5520

1 Microarchitecture

Von Neumann architecture

2 Instruction Memory and Data Memory Size

1024 x 16-bit

3 Total Number of Registers

16 Registers

4 Instruction Format

Instruction Set 2

The instructions are 16-bit each divided into 4 types

1-Arithmetic:

Sub,add,mult,or

2 bits TYPE	3 bits CODE	3 bits 1st OPER	3 bits 2nd OPER	2bits UNUSED	3 bits DESTIN

Type: is generic for all types of instructions

00	Arithmetic
01	Immediate
10	Branch
11	Shift

Code:3-bit to identify the specific instruction operation , unique for each type

000	SUB
001	ADD
010	MULT
011	OR

1st Oper & 2nd Oper and dest: 3bit to specify the address of the required register which will be one of the 16 16bit general-purpose registers.

Ex.SUB->dest=1st oper-2nd oper

2-Immediate:

Addi,andi,lw,sw,slti

2 bits TYPE	3 bits CODE	3 bits 1st OPER	3 bits 2nd OPER	5 bits IMMED

Type: is generic for all types of instructions

Code:3-bit to identify the specific instruction operation , unique for each type

000	ADDI
001	SLTI
010	ANDI
011	LW
100	SW

1st Oper & 2nd Oper: 3bit to specify the address of the required register which will be one of the 16 16bit general-purpose registers.

Ex.ADDI->2nd oper=1st oper+immed

<u>3-Branch & Jump:</u>

Beq,blt

2 bits TYPE	3 bits CODE	3 bits 1st OPER	3 bits 2nd OPER	5 bits LABEL

Jump

2 bits TYPE	3 bits CODE	11 bits ADDRESS

Type: is generic for all types of instructions

Code:3-bit to identify the specific instruction operation , unique for each type

000	BEQ
001	BLT
010	J

1st Oper & 2nd Oper: 3bit to specify the address of the required register which will be one of the 16 16bit general-purpose registers.

Ex.BLT->1st oper<2nd oper ,then pc=label

4-Shift:

Sll,srl

2 bits TYPE	3 bits CODE	3 bits 1st OPER	5 bits SHIFTVALUE	3 bits DEST

Type: is generic for all types of instructions

Code:3-bit to identify the specific instruction operation , unique for each type

000	SLL
001	SLR

1st Oper & dest: 3bit to specify the address of the required register which will be one of the 16 16bit general-purpose registers.

Ex.sll->dest=1st oper << shiftvalue

GENERAL EXAMPLES/TEST CASES:

1-Add: 0000100000100011

2-Addi: 0100000000100001

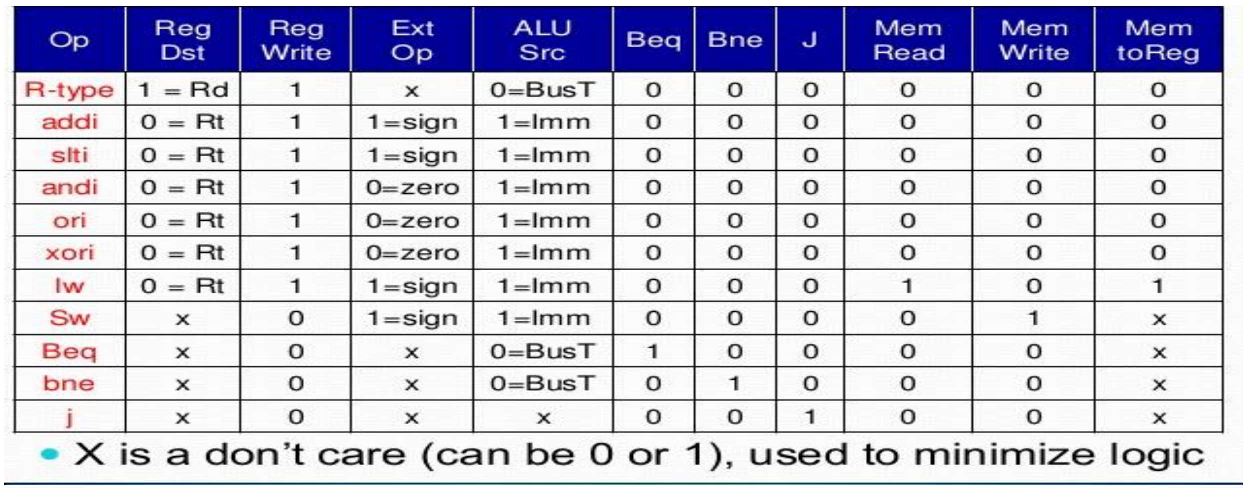
3-Beq: 1000000000100000

4-sll: 1100000000011111

5-jump: 1001000001110000

ALUCONTROL SIGNALS:

Instruction	ALUop	RegDst	ALUSrc	RegWrite	MemRead	MemWrite	Branch	MemtoReg
LW	00	0	1	1	1	0	0	1
SW	00	X	1	0	0	1	0	X
BEQ	01	X	0	0	0	0	1	X
R-TYPE	10	1	0	1	0	0	0	0



Op	Reg Dst	Reg Write	Ext Op	ALU Src	Beq	Bne	J	Mem Read	Mem Write	Mem toReg
R-type	1 = Rd	1	x	0=BusT	0	0	0	0	0	0
addi	0 = Rt	1	1=sign	1=Imm	0	0	0	0	0	0
slti	0 = Rt	1	1=sign	1=Imm	0	0	0	0	0	0
andi	0 = Rt	1	0=zero	1=Imm	0	0	0	0	0	0
ori	0 = Rt	1	0=zero	1=Imm	0	0	0	0	0	0
xori	0 = Rt	1	0=zero	1=Imm	0	0	0	0	0	0
lw	0 = Rt	1	1=sign	1=Imm	0	0	0	1	0	1
Sw	x	0	1=sign	1=Imm	0	0	0	0	1	x
Beq	x	0	x	0=BusT	1	0	0	0	0	x
bne	x	0	x	0=BusT	0	1	0	0	0	x
j	x	0	x	x	0	0	1	0	0	x

- X is a don't care (can be 0 or 1), used to minimize logic

Signal Name	Effect when not set	Effect when set
RegDst	Destination register comes from rt field.	Destination register comes from the rd field.
RegWrite	None.	Write register is written to with Write Data.
ALUSrc	Second ALU operand is Read Data 2.	Second ALU operand is immediate field.
PCSrc	PC ->PC + 4	PC ->Branch target
MemRead	None.	Contents of address input are copied to Read Data.
MemWrite	None.	Write Data is written to Address
MemToReg	Value of register Write Data is from ALU	Value of register Write Data is memory Read Data.
jump	Does not have a jump address	Has a jump address in the last 11 bits
shift	Does not have a shift value	Has a shift amount in the ***shiftvalue*** field

ALUOp 2-bit CONTROL: identical to the ***Type*** field in each instruction.

ALU Operation 4-bit Input:

0000	AND
0001	OR
0010	ADD
0110	SUB
0111	SLT
1110	SLL
1111	SRL
1100	MULT

<u>Datapath:</u>

1-The Datapath does not differ from MIPS except for the InstructionDecode parts (the divided Instruction) to fit the previously mentioned fields as well as the addresses for the Jump and Branch to fi the 16-bit approach of the instruction and address sizes. Also, we added a signal for the ***shift*** and ***jump*** Which will be set if the instruction is shift or jump respectively.The instruction is first fetched in the fetch class then decoded in the deocde class according to the type to be assigned in the readdata1,readdata2 field and writeRegister and shift amount in case of shift and assign the 10 control signals described above as well as the aluop 2 bit control which is the 2bit ***type*** field in the beginning of each instruction.The execute class then is responsible for the ALUcontrol unit which generates the ALU operation 4bit control from the ***type*** field as well as the ***code*** field and then the operation is computed in the evaluator method.The memoryAccess class which is responsible for accessing the cache which is directmapped so doesn't have replacement policy its built-in with a size of 16 blocks.It also updates the data/instruction memory since our architecture is von neumann with the value of the readdata2 depending on the control signals(memread and memwrite) and the PC value is also assigned according to the zeroflag,negflag generated from the execute class and the branch,jump signal whether it will have the value of the branchAddress or JumpAddress calculated by the execute class as well.Finally the writeback class which writes back the values in the cache as well as the memory according to the memtoreg and memwrite signals.For the pipelining we implemented 4 classes of registers (IF,EXC,MEM,WB) each of them respectively responsible for carrying the values/outputs of each stage to be used in the coming stage of each class.These registers consist of multiple of static variables to hold the inputs needed for the next stage of each instruction which after being executed these registered are cleared and ready to be occupied by the next instruction.Unfortunately we couldn’t work the code to run the stages all pipelined concurrently instead they will be working sequentially with values in each register displayed in each cycle.Here are the outputs or running the above test cases in class ***CPU.***After also loading the instructions in the instruction memory and cache.