ASSIGNMENT - 07

CS220

PREPARED BY :

CHIRAYUSH MOHANTY

210289

Y21 UG CSE

mailto:cmohanty21@iitk.ac.in

PROBLEM STATEMENT:

We want to create a new processor CSE-BUBBLE that has the instruction set architecture as shown in Tab.1 below. We assume that the processor word size and instruction size are 32 bits and we use the VEDA memory as implemented in Assignment 4. The VEDA memory has two parts: a) Instruction Memory, b) Data Memory. You have to make sure that these two sections do not overlap. We also assume that RISC-BUBBLE has total 32 registers of which certain registers follow the same roles as in MIPS-32 ISA. For example, Program Counter (PC) register holds the address of the next instruction. The target is to create the op-code formats for the given ISA, shown in Tables below, decide upon the data path elements (such as addition, subtraction, shift, jump) and then develop different verilog modules for the same along with the finite state machine for the control path. Finally we shall build a single-cycle instruction execution unit for CSE-BUBBLE.

INSTRUCTION SET FOR CSE-BUBBLE:

1) Arithmetic:

Instruction	Meaning
add r0, r1, r2	r0=r1+r2
sub r0, r1, r2	r0=r1-r2
addu r0, r1, r2	r0=r1+r2 (unsigned addition, not 2's complement)
subu r0,r1,r2	r0=r1-r2 (unsigned addition, not 2's complement)
addi r0,r1,1000	r0=r1+1000
addiu r0,r1, 1000	r0=r1+1000 (unsigned additon, not 2's complement)

2)Logical:

and r0,r1,r2	r0= r1 & r2
or r0,r1,r2	r0=r1 | r2
andi r0,r1, 1000	r0= r1 & 1000
ori r0,r1, 1000	r0= r1 | 1000

sll r0, r1, 10	r0=r1<<10 (shift left logical)
srl r0, r1, 10	r0=r1>>10 (shift right logical)

3)Data transfer:

lw r0,10(r1)	r0=Memory[r1+10] (load word)
sw r0,10(r1)	Memory[r1+10]=r0 (store word)

4) Conditional Branch:

beq r0,r1,10	if(r0==r1) go to PC+4+10 (branch on equal)
bne r0,r1,10	if(r0!=r1) go to PC+4+10 (branch on not equal)
bgt r0,r1,10	if(r0>r1) go to PC+4+10 (branch if greater than)
bgte r0,r1, 10	if(r0>=r1) go to PC+4+10 (branch if greter than or equal)
ble r0,r1, 10	if(r0<r1) go to PC+4+10 (branch if less than)
bleq r0,r1, 10	if(r0<=r1) go to PC+4+10 (branch if less than or equal)

5)Unconditional Branch:

j 10	jump to address 10
jr r0	jump to address stored in register r0
jr r0	ra=PC+4 and jump to address 10

6)Comparison:

slt r0,r1,r2	if(r1<r2) r0=1 else r0=0
slti 1,2,100	if(r1<100) r0=1 else r0=0

Process Development Strategy:

Based on the provided instruction set for CSE_BUBBLE processor,the following data path elements can be identified:

1.Register File:The processor has a total of 32 registers, which can be used for storing and performing arithmetic and logical operations
2.ALU(Arithmetic and Logic Unit): The ALU is responsible for performing arithmetic and logical operations such as addition , subtraction, AND,OR,etc on data stored in registers.
3.Instruction Memory:The instruction memory hold the instructions to be executed by the processor.It fetches the instructions based on the address proioivided by the program counter(PC).
4.Data Memory:The data memory is used for storing data that is loaded from or stored to memory by the processor instructions, such as load word(lw) and store word(sw).
5.PC(Program Counter): The PC holds the address of the next instruction to be fetched from the instruction memory.
6.Sign Extender :The sign extender is used to extend the sign bit of an immediate value used in arithmetic or logical operations.
7.Shifters: The processor supports logical left shift and right operations which require shifters to shift the bits of a register by a certain number of positions.
8.Comparator: The comparator is used to compare the values of two registers and determine the result of conditional branch instructions, such as beq, bne, bgt, etc.
9.MUX (Multiplexer): The MUX is used to select between different sources of input data, such as selecting between the output of the ALU and the result of a load instruction to be written to a register
10.Control Signals: The processor requires various control signals to enable or disable different data path elements and control the flow of data and instructions within the processor.

1.[PDS1] Decide the registers and their usage protocol.
Solution.

For the CSE-BUBBLE processor, we assume a total of 32 registers, each of 32-bit size. The registers are numbered from 0 to 31. The following registers are used for specific purposes:

->$zero (0): This register always holds the value 0.
->$at (1): This register is reserved for the assembler and is not used by the programmer.
->$v0-$v1 (2-3): These registers are used to hold the return values from a subroutine.
->$a0-$a3 (4-7): These registers are used to pass arguments to a subroutine.
->$t0-$t7 (8-15): These registers are used for temporary storage.
->$s0-$s7 (16-23): These registers are used for saving values that need to be preserved across subroutine calls.
-> $t8-$t9 (24-25): These registers are used for temporary storage.
-> $k0-$k1 (26-27): These registers are reserved for use by the operating system kernel.
-> $gp (28): This register is used as a global pointer.
-> $sp (29): This register is used as a stack pointer.
-> $fp (30): This register is used as a frame pointer.
-> $ra (31): This register is used to hold the return address from a subroutine.

In addition to these registers, the PC (program counter) register is used to hold the address of the next instruction to be executed.
The usage protocol for the registers is as follows:

-> The registers $zero, $at, $v0-$v1, and $a0-$a3 are caller-saved, meaning that they can be modified by a subroutine and the changes will not be preserved across a subroutine call.
-> The registers $t0-$t9 and $s0-$s7 are callee-saved, meaning that they must be preserved across subroutine calls.The subroutine must save these registers on the stack if they are modified and restore them before returning.
-> The registers $k0-$k1 are reserved for use by the operating system kernel and should not be modified by user programs.
->The registers $gp, $sp, $fp, and $ra are used for specific purposes and their usage is prescribed by the instruction set architecture.

2.[PDS2] Decide upon the size for instruction and data memory in VEDA.
Solution.

As per the problem statement, the processor word size and instruction size are 32 bits. So, we need to have an instruction memory of at least 2^32 bytes. However, as we do not need to use the entire memory, we can choose a smaller size.

For the data memory, we need to decide upon the amount of memory required to store data. This depends on the application that the processor is intended for. Since there is no information given about the specific application, we can assume a moderate size of 2^16 bytes for data memory.

Therefore, we can decide to have an instruction memory of 64KB (2^16 words), and a data memory of 64KB (2^16 bytes).

3.[PDS3] Design the instruction layout for R-, I- and J-type instructions and their respective encoding methodologies.
Solution.

To design the instruction layout for CSE-BUBBLE, we can follow the traditional MIPS-style instruction format, which has the following three types of instructions: R-type, I-type, and J-type.

R-type instructions:

The R-type instructions operate on registers only. The instruction format for R-type instructions is as follows:

6-bit op	5-bit rs	5-bit rt	5-bit rd	5-bit shamt	6-bit funct
opcode	source	source	dest	shift amount	function code

where,
op: 6-bit opcode field
rs: 5-bit source register field
rt: 5-bit source register field
rd: 5-bit destination register field
shamt: 5-bit shift amount field
funct: 6-bit function code field

I-type instructions:

The I-type instructions operate on immediate values and registers. The instruction format for I-type instructions is as follows:

6-bit op	5-bit rs	5-bit rt	16-bit immediate

Opcode source dest immediate

where,
op: 6-bit opcode field
rs: 5-bit source register field
rt: 5-bit destination register field
immediate: 16-bit immediate value

J-type instructions:
The J-type instructions operate on jump targets. The instruction format for J-type instructions is as follows:

6-bit op 26-bit address
opcode target address

where,
op: 6-bit opcode field
address: 26-bit jump target address

For encoding these instructions, we can use the following methodology:

R-type instructions:
->The opcode field for all R-type instructions can be set to 0b000000. The funct field will be different for each R-type instruction.
->The opcode and funct fields can be used to determine the specific operation to be performed by the ALU.
I-type instructions:
->The opcode field for all I-type instructions can be assigned with unique opcodes. For example, we can use 0b001000 for addi instruction,0b001001 for addiu instruction, 0b001101 for ori instruction, and so on.

J-type instructions:
->The opcode field for all J-type instructions can be assigned with unique opcodes. For example, we can use 0b000010 for j instruction, 0b000011 for jal instruction, and so on.

Overall, the instruction layout and encoding methodology for CSE-BUBBLE can be implemented as follows:

R-type Instructions:

The opcode for all R-type instructions will be 000000.
The funct field will specify the particular R-type instruction. The encoding for the funct field will be as follows:
add instruction: 100000
sub instruction: 100010
addu instruction: 100001
subu instruction: 100011
and instruction: 100100
or instruction: 100101
sll instruction: 000000
srl instruction: 000010
jr instruction: 001000
slt instruction: 101010

I-type Instructions:

The encoding for the opcode field will be as follows:
addi instruction: 001000
addiu instruction: 001001
andi instruction: 001100
ori instruction: 001101
lw instruction: 100011
sw instruction: 101011
beq instruction: 000100
bne instruction: 000101
slti instruction: 001010

J-type Instructions:

The opcode for all J-type instructions will be 000010.
Note: The rs, rt, rd, and shamt fields in the R-type instruction format will specify the registers that are involved in the operation,while the immediate field in the I-type instruction format will specify the immediate value used in the instruction.The address field in the J-type instruction format will specify the address to jump to.

4.[PDS4] Now design and implement an instruction fetch phase where the instruction next to be executed will be stored in the instruction register.
Solution.

For this I have made two verilog modules: instructon_fetch.v and instruction_fetch_tb.v as the test bench module for instruction_fetch.v.

```
In the module of Instruction_fetch, we use the VEDA module in read mode
(mode input set to 1'b1) to read the instruction at the address specified
by the upper 28 bits of the PC register (address wire). We then output the
instruction value to the instruction output wire.

Note that we assume that the instruction memory is mapped to the upper 1GB
of the 4 GB address space, so we only use the upper 28 bits of the PC
register to calculate the instruction memory address. Also, we assume that
the instruction memory is byte-addressable, so we don't need to shift the
address left by 2 bits to account for the word size. Finally, we assume
that the instruction memory is already populated with the machine code for
the program to be executed.
```

5.[PDS5]Next design and implement a module for instruction decode to identify which data path element to execute given the opcode of the instruction.
Solution.

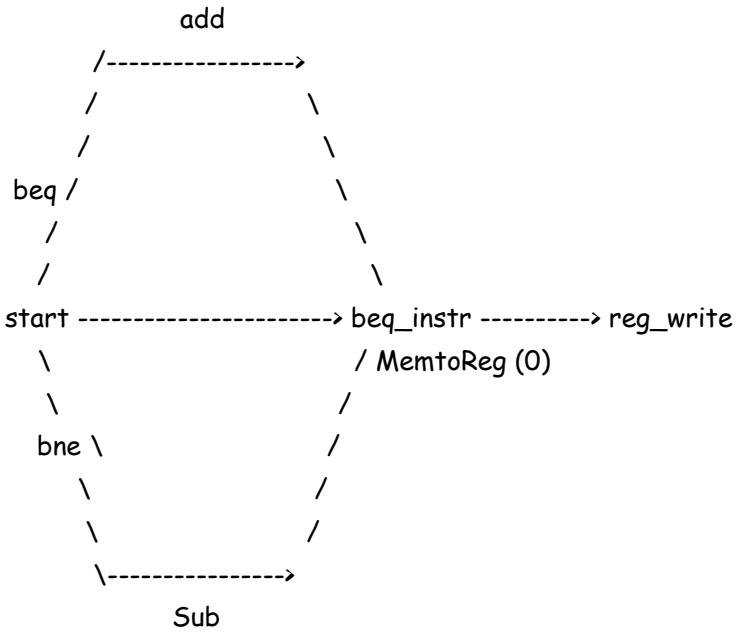
For this I have made two verilog modules : instruction_decode.v and instruction_decode_tb.v to implement the instruction decoding and perform the respective operations. In the instruction_decode.v module I have properly commented all the decodings and have also added assert statements for exception handling incase we give some unknown instruction that is not previously defined. In the test bench module I have tested different instruction decodings like R-type instruction testing, I-type instruction testing , conditional branch testing,etc.

6.[PDS6] + [PDS7] Design and implement the Arithmetic Logic Unit (ALU) in top-down approach to develop different modules for different types of instructions. There might be some hardware parts in the ALU that can be shared by different modules if required. This will reduce the footprint of the hardware.
Solution.

For this I have made the verilog module named as alu.v along with the test bench file alu_tb.v. In the file alu.v the instruction_fetch.v module is also included along with the new creation of a module named as control for appropriate implementation. Also have included the alu branching operations in the same module as well for alu implementation.

8.[PDS8] Design the finite state machine for the control signals to execute the processor. Please ensure that every instruction should be executed in single clock cycle. Finally write the test benches to simulate the CSE-BUBBLE.

For this I have made the fsm.v file which contains the fsm module as per the control signals to execute the processor and also fsm_tb.v file as the test bench module for the fsm module that we talked about earlier. Along with elaborate testing it also has assert statements to take care of the exceptions in the test bench instructions. The fsm state diagram that we obtained is :



9.[PDS9] + . [PDS10]
Solution.

I have written the MIPS code for bubble sort in the file named as "bubblesort.asm" which when run in QTSpim sorts a given array!! Further in the file "final.asm", I have converted this into machine code following the given ISA in our problem , then we store the machine code in the instruction memory of Veda and execute the output of the bubble sort in the data memory which can be further accessed from tour defined memory.