

ISTANBUL TECHNICAL UNIVERSITY
COMPUTER ENGINEERING DEPARTMENT

BLG 222E
COMPUTER ORGANIZATION
PROJECT REPORT

PROJECT NO : 1
PROJECT DATE : 11.05.2021
CLASS SESSION : THURSDAY - 08.30
GROUP NO : 4

GROUP MEMBERS:

070190110 : MERT & BOZKURTLAR
070200734 : MUHAMMED NECATİ & POLAT
150180252 : CANER & COŞKUN (Representative)

SPRING 2022

Contents

FRONT COVER

CONTENTS

1	INTRODUCTION [10 points]	1
2	MATERIALS AND METHODS [40 points]	1
2.1	Hardwired Control Unit	1
2.1.1	Instruction Fetching and Decoding	1
2.1.2	Memory Reference Instructions	2
2.1.3	Register Reference Instructions	2
2.2	Basic Computer	3
2.3	Example Code	3
3	RESULTS [15 points]	4
4	DISCUSSION [25 points]	5
5	CONCLUSION [10 points]	6

1 INTRODUCTION [10 points]

In this homework it is expected from us to implement a basic computer using verilog. We will design a Hardwired Control Unit (HCU) and then we will connect Hardwired Control Unit and ALU System together. HCU will take inputs from the ALU system and will drive it accordingly. In this computer, we are using a 8 bit RAM so we will load IR in 2 cycle and decode it. Then our system will act accordingly to the opcode given in the IR. After implementing the HCU we will connect it with the ALU system in a module called Basic Computer. Then we are going to write simulations and show the results.

2 MATERIALS AND METHODS [40 points]

2.1 Hardwired Control Unit

In this module the inputs are clock, reset, IROut(instruction) and ALUOutFlag(required for BNE instruction). Outputs are mostly inputs of the ALU System; MemWR, MemCS, Out selections of Register File (RF) and Address Register File (ARF) and FunSel's of RF, ARF, ALU and IR, then selection inputs of MuxASel, MuxBSel, MuxCSel. Enable (RegSel for ARF and RF) inputs of RF, ARF, Enable and L/H inputs for IR. Then we defined wires to hold decoded parts of the IR such as opcode, destreg, srcreg1, srcreg2, memrefop, TimerFunSel, regsel, TimerOut(these wires are given as output, but this is for debug only and they are not needed to be defined as outputs). Our HCU implementation uses a timer module, which is a 4 bit clock ticking at positive edge of Clock if *TimerFunSel* = 0 and clearing its value to 0 if *TimerFunSel* = 1. Assignments of opcode, destreg, srcreg1, srcreg2, memrefop are done outside the always block since they directly use IROut as a wire.

HCU decides which flags to set in an Always block, if *reset* = 1, then all registers(RF, ARF, IR) are cleared with setting their respective flags and enabling all of their registers. If *reset* = 0, then HCU decides its operation based on current timer and opcode. If the instruction is completed, then timer and IR should be cleared.

2.1.1 Instruction Fetching and Decoding

Instruction fetching and decoding is done in 2 cycles, both cycles are used to fetch the instruction from the memory since an instruction is 16 bit but our Memory module is 8 bit. At the first cycle(*t0*), IR is enabled and set to load mode, and Lower 8 bits of the IR is loaded from memory since instructions are stored in little endian order. The second cycle(*t1*) is the same as first, except Higher 8 bits of the IR is loaded from

memory. Decoding of instructions doesn't need an extra cycle since required wires(opcode, destreg etc.) are directly connected to IROut and we can access the memory whenever required from any register in ARF. After the fetching and decoding, HCU starts the loaded instruction's operations in the next clock(t2).

2.1.2 Memory Reference Instructions

If opcode is 0, 1, 2, B, C or F then the instruction is a memory reference instruction.

In op 0 & t2, PC is loaded with IR Low and the instruction ends(clear timer and IR).

In op 1 & t2, selected register is enabled and set to load mode, and depending on the addressing of instruction, MuxA selects IR Low or MemoryOut with address AR and the instruction ends.

In op 2 & t2, selected register is given to ALU and Write to memory is enabled, address of the write is AR and the instruction ends.

In op B & t2, SP is enabled and set to increment and given as address to the Memory. Memory read is enabled and selected register is enabled and set to load mode and the instruction ends. Note that current SP increases after reading.

In op C & t2, SP is enabled and set to decrement and given as address to the Memory. Memory write is enabled and selected register which comes through ALU is written to Memory, and the instruction ends. Note that current SP decreases after the write.

In op F & t3, if the Z flag of ALUOutFlag is 0, PC is loaded with IR Low, if not the instruction does nothing. Instruction ends in both cases.

2.1.3 Register Reference Instructions

If the instruction is not a memory reference instruction then usage of ALU is needed and a destreg should be enabled. In t2 we will enable the destreg and set its function to load. Then we will choose outputs of RF, ARF, MuxCSel according to the srcreg1 and srcreg2, after selecting srcregs and destreg we will determine the FunSel of the ALU according to the opcode. D and E Instructions use ALU only as a connection point to load the destreg with given srcreg1. All of the register reference instructions will be completed after that step except op 8, D and E. Opcode 8 wants from us srcreg2 - srcreg1 but we can not calculate it within a clock due the limitations of our system so we will calculate it by taking 2's complement of srcreg1-srcreg2, which can be calculated directly. We have to first calculate (t2) srcreg1-srcreg2 and load it to the destreg, (t3) then NOT(destreg) and load it to the destreg again, (t4)at last we will increment destreg. D and E operations also need increment and decrement so we should check if the opcode is D or E and if we are at t3, if so we will increment or decrement according to input. After that all register reference instructions is complete. Except D, E, or 8 we can also reset timer and IR

inside the if blocks that we selected ALU's FunSel because operations can be done in 1 cycle but in D E and 8 we should wait until they finished and we will reset them. Also 8, D and E instructions give their results to the ALU at their last step to get the correct flags from the results.

2.2 Basic Computer

In this module the inputs are clock and reset. Then we used modules ALU System and Hardwired Control Unit in this module. Input and Output connections of ALU System and HCU is done in this module, HCU's outputs are mostly used as ALU System's inputs and IROut and ALUOutFlag outputs of ALU System is used as inputs for HCU.

2.3 Example Code

The code to be executed is written in the ram in hexadecimal numbers in little endian. It calculates the total of numbers at $M[A0]$ to $M[A4]$, then stores the result in $M[A6]$.

```
01 0x00: BRA 0x20 = 04 20

33 0x20: LD R1 IM 0x05 = 14 05
35 0x22: LD R2 IM 0x00 = 15 00
37 0x24: LD R3 IM 0xA0 = 16 A0
39 0x26: MOV AR R3 = 32 60
41 0x28: LABEL: LD R3 D = 12 00
43 0x2A: ADD R2 R2 R3 = 75 56
45 0x2C: INC AR AR = D2 20
47 0x2E: DEC R1 R1 = E4 40
49 0x30: BNE IM LABEL = F4 28 //28 in decimal is 40(41st line)
51 0x32: INC AR AR = D2 20
53 0x34: ST R2 D = 21 00
//end
55 0x36: BRA 0x36 = 04 36
```

The code is written in little endian in the RAM file(for example, for instruction 1405, 05 is written at 0x20 and 14 is written in 0x21). The code completes in 1085ns in our implementation. The last line is to branch to the end indefinitely since the execution is complete and the code shouldn't repeat.

3 RESULTS [15 points]

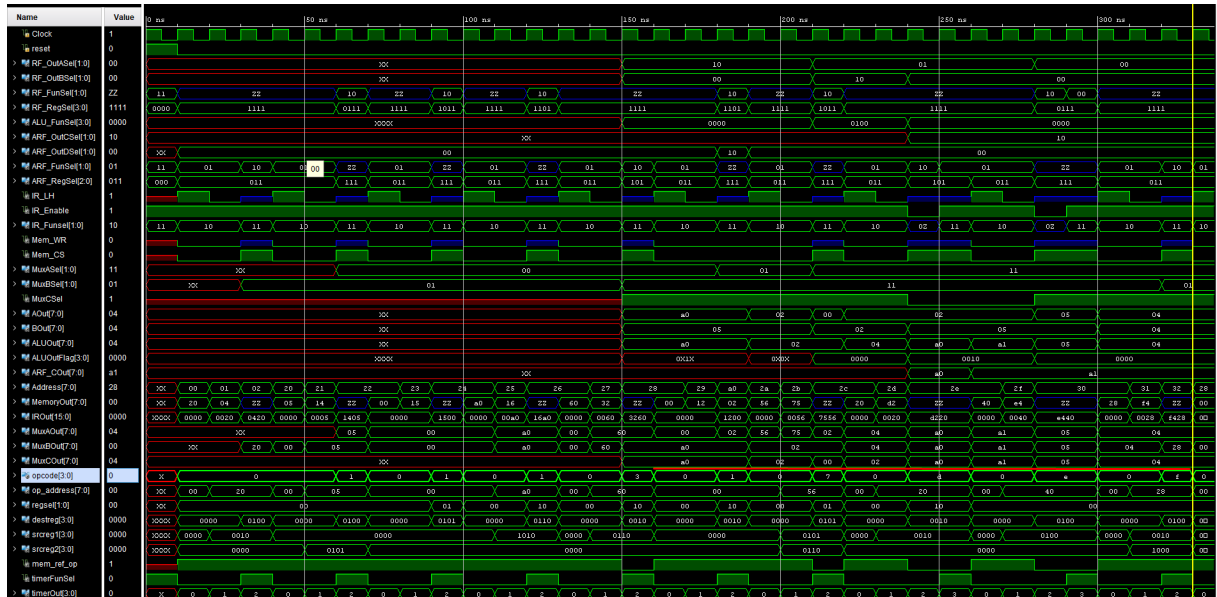


Figure 1: Start and first loop of the code, first loop is shown with a red line

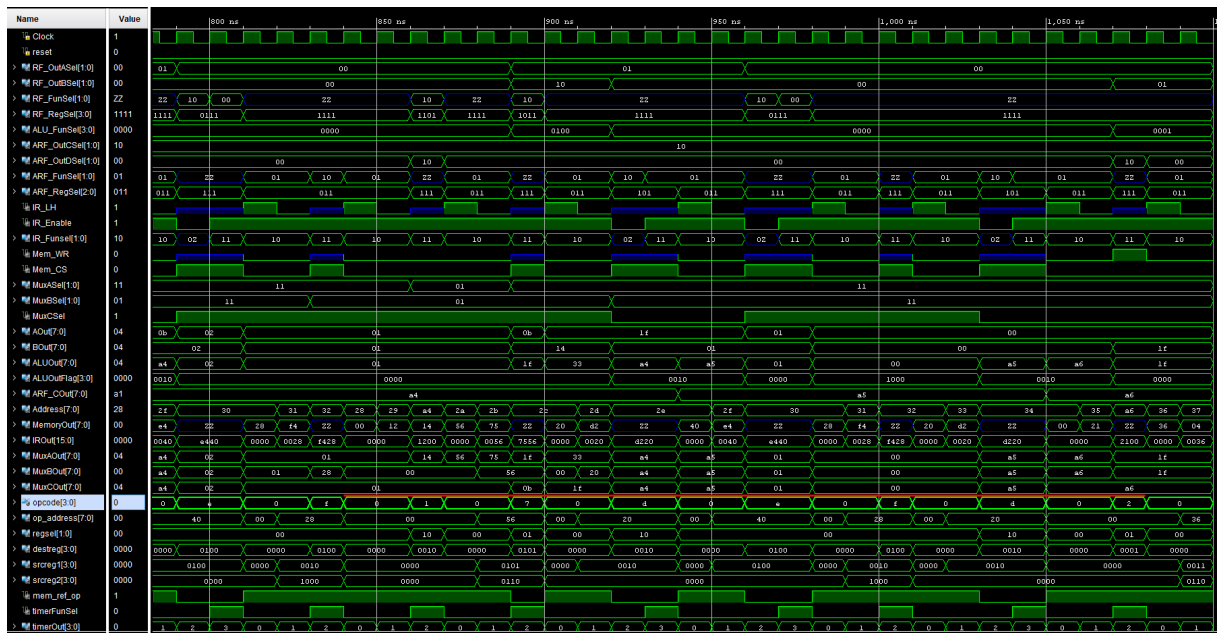


Figure 2: End of the execution, shown with a red line

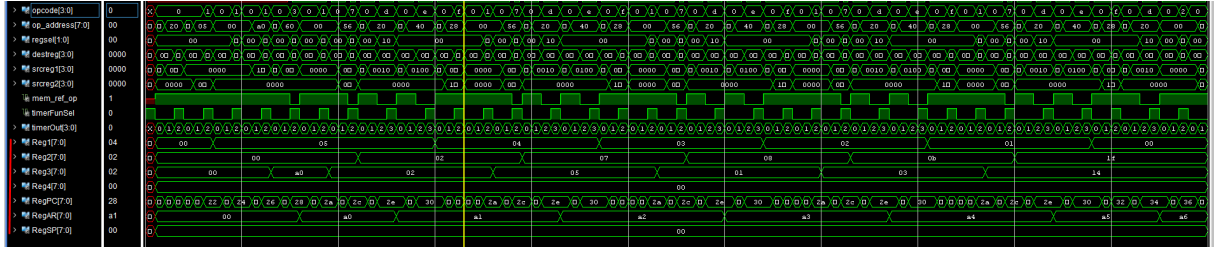


Figure 3: Register values for the whole execution








>  [160][7:0]	02	Array
>  [161][7:0]	05	Array
>  [162][7:0]	01	Array
>  [163][7:0]	03	Array
>  [164][7:0]	14	Array
>  [165][7:0]	00	Array
>  [166][7:0]	1f	Array

Figure 4: Memory at the end of the execution

4 DISCUSSION [25 points]

At the start of the simulation, reset input is given, which results in clearing RF, ARF, IR. Then, at the start the first instruction is loaded from memory, which is BRA 0x20 = 0420. It is loaded as little endian so first the lower half(20) loads then at the next clock, the higher half loads(04). At the last step it loads 20 to PC, which is branching to 0x20.

Then required values are loaded to respective registers in the next 3 instructions. R1 is used as iteration counter, R2 is used to store the total so far, R3 is used to get the values in memory. Then AR is set to A0 which was loaded to R3 previously. Then the loop starts, first R3 is loaded from memory, so $R3 < -M[AR = A0]$ (values in memory can be seen in 4). Then R3 is added to R2 and stored in R2, then AR is increased and R1

is decreased. At the end of the loop, it goes back to the start of the loop if $R1 \neq 0$ ($Z = 0$ since last operation is done on R1). The loop executes for five times (until $R1=0$), then AR is increased again (A5 at the end of loop), then R2(total) is stored in $M[AR = A6]$.

As it can be seen in 4, total of values given in $M[A0 = 160]$ to $M[A4 = 164]$ is stored in $M[A6 = 166]$ at the end.

5 CONCLUSION [10 points]

In conclusion, we wrote a Basic Computer that can do calculations on 8 bit numbers with a 256 byte memory including the code. We learned the architecture of a hardware controlled basic computer and created our own in according to the given design choices and constraints.

It is worth noting that our computer can't do arithmetic on numbers bigger than 8 bits with current instructions since we don't use carry flag of the ALU in any of the instructions.