Assignment 1

Computer Organization and Architecture

Aditya Choudhary | 20CS10005

Questions

1. Study the one instruction CPU ISAs and select one.
2. Code one multiplication or division routine using the chosen instruction.
3. Design the CPU data paths for your one instruction CPU.
4. Develop the controller specifications to orchestrate the data path components so that the required instruction is properly executed.

Answer:

1. **Subtract and branch if negative**
   The subneg a,b,L instruction subtracts the contents at address a from the contents at address b and stores the result at address b, and then if the result is negative it transfers the control to address L. If the result is not negative then execution proceeds to the next address in instructions.
   We are developing a Harward Architecture for the CPU, hence our instruction and data memory are going to be different. This means that a and b are address in the data memory and L is the address in the instruction memory which the program counter is supposed to point at.
   Pseudo code

```
subneg a,b,L
data_memory[b]=data_memory[b]-data_memory[a]
if(data_memory[b]<0)
    goto L (in instruction memory)
```

   To avoid the conditional branching, we can pass the next address in place of L in the instruction, this means that even if branching has to take place, it will only jump to the next instruction in the sequence. We pass the next address in the instruction if there are only two operands in the instruction.

2. **Coding Division and Multiplication using subneg**
   We can implement division in subneg ISA using the repeated subtraction method. We will subtract divisor from the divident until the divident becomes less than zero. The number of times we have to subtract divisor will become the quotient and divident when added to divisor will give us the remainder.

   Whenever we use two operands in the instruction, we mean that we are suppressing conditional branch, and the L field in the instruction contains the address of the next instruction in the sequence. Data memory locations Z and Z1 contain constants 0 and 1 respectively. Divisor is contained at memory location a and divident is contained at memory location b. Quotient will be contained at memory location q and remainder will be contained at location r.

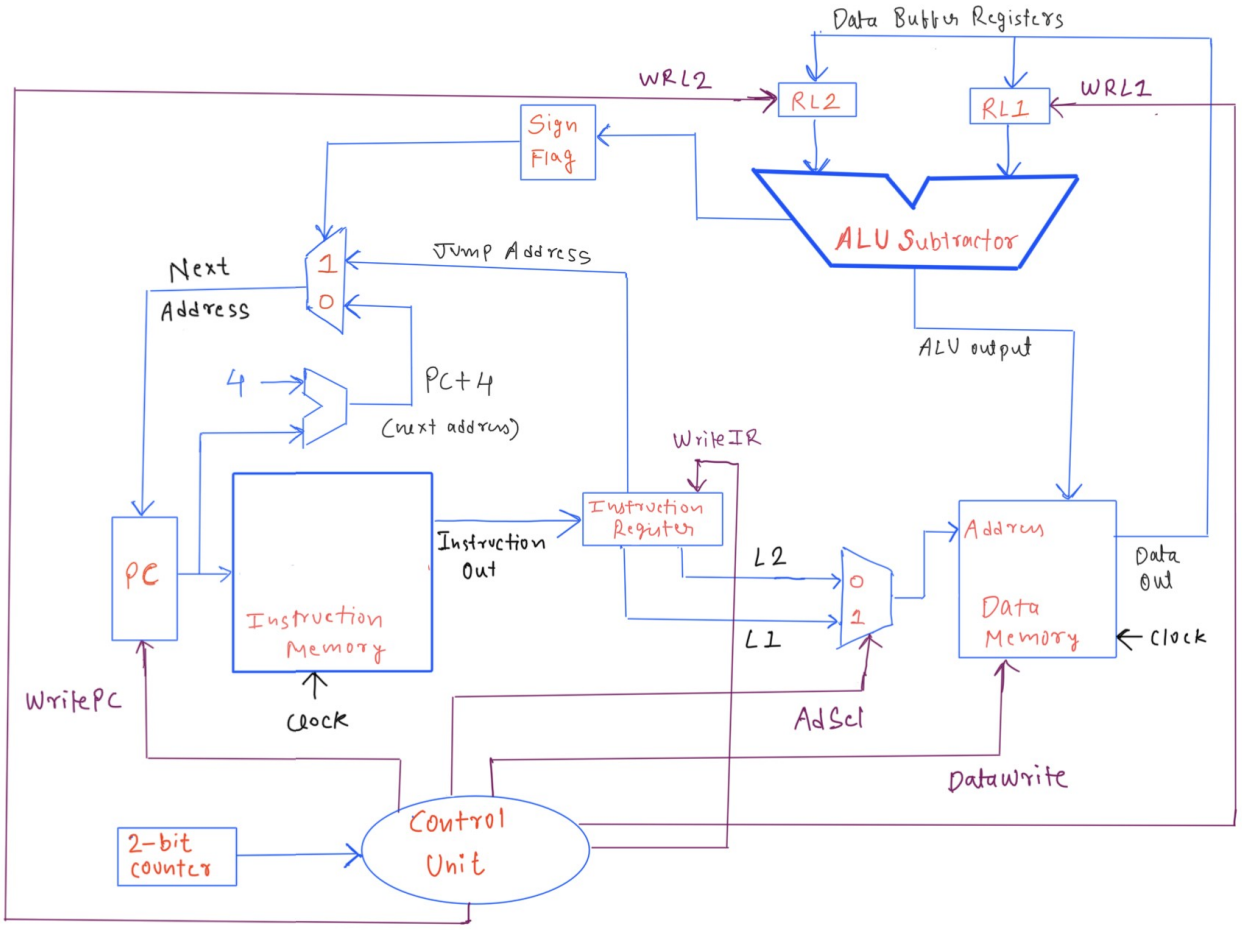
```
START:
    subneg q,q              # clearing q, mem[q] = 0
LOOP:
    subneg  Z,Z             # clearing Z, mem[Z] = 0
    subneg  a,b,FINISH      # mem[b]=mem[b]-mem[a] (if(mem[b]<0) goto FINSH)
    subneg  Z1,Z            # mem[Z]=mem[Z]-mem[Z1] (mem[Z] = -1)
    subneg  Z,q             # mem[q]=mem[q]-mem[Z] (mem[q]=mem[q]+1)
    subneg  Z,Z             # restoring Z, mem[Z] = 0
    subneg  Z1,Z,LOOP       # mem[Z] = -1, therefore will always branch
FINISH:
    subneg a,Z              # mem[b] = mem[b] + mem[a]
    subneg Z,b              # restoring remainder in b
    subneg Z,Z
                            # moving remainder from b to r
    subneg r,r
    subneg b,Z
    subneg Z,r
    subneg Z,Z
```

   Multiplication can be similarly implemented. Suppose we have to multiply two positive integers r and s, we loop through r times, adding s to a memory location p initialised to 0 in every iteration. The data memory location p will thus contain the multiplication of r and s.

   Data memory locations Z and Z1 contain constants 0 and 1 respectively. Multiplier is contained at memory location a and Multiplicand is contained at memory location b. Product will be contained in memory location p.

```
START:
    subneg p,p              # clearing p, mem[p] = 0
LOOP:
    subneg Z,Z              # restoring Z, mem[Z] = 0
    subneg Z1,b,FINISH      # mem[b] = mem[b]-1, (if(mem[b] < 0) goto FINISH)
    subneg a,Z              # mem[Z] = -mem[a]
    subneg Z,p              # mem[p] = mem[p] - mem[Z], mem[p] += mem[a]
    subneg Z,Z              # restoring Z, mem[Z] = 0
    subneg Z1,Z,LOOP        # mem[Z] = -1, therefore will always branch
FINISH:                     # final product at location p
```

3. **Datapath Design for executing subneg instruction**



Instruction Format

L1	L2	Jump Addr.

In this datapath, the CPU is based on multicycle Harvard Architecture. The instructions loaded in the instruction memory are loaded in the Instruction register. From there, the memory addresses L1 and L2 are given to the data memory through a MUX. We use a MUX as we can access the memory location of only one location at a time.

We then load the data at the memory locations in the data buffer registers, from there the ALU outputs the subtracted result. The result is given back to the data memory and the memory location L2 is updated. The ALU also sets up the sign flag after computation, which decides the next address which the program counter will point to.

If sign flag is 1, we want the CPU to jump to the jump address, therefore we pass PC+4 and the jump address through a MUX.

For all these operations, we generate appropriate control signals at appropriate value of the 2-bit counter attached to the Control Unit.

4. **Develop the controller specifications to orchestrate the data path components so that the required instruction is properly executed**

   In our multicycle control unit design, we have used a 2-bit counter as a state machine for the counter to decide the output of control signals at varipus stages of the fetch-execution cycle. When counter is 00, WriteIR control signal is set to 1. This is the fetch phase. When counter becomes 01, data at memory location L2 is stored in RL2 register as MUX selects L2 address and WRL2 is set to 1. In the next cycle, L1 address content is loaded in RL1 register and ALU result as well as the sign flag is computed. Then, when is the counter is 11, WritePC and DataWrite are set to 1, enabling data memory to write new value at location L2 and program counter pointing to the next address in the instruction memory. Counter then becomes 00 and this sequence continues.

   This is the table of the counter value and the various control signals.

CounterVal	WriteIR	AdSel	WRL1	WRL2	DataWrite	WritePC
00	1	0	0	0	0	0
01	0	0	0	1	0	0
10	0	1	1	0	0	0
11	0	0	0	0	1	1