Computer Organisation and Architecture

CS31007
Assignment-1 Report

Ashwani Kumar Kamal (20CS10011)

October 31, 2022

---

1 Problem Statement

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.

1.1 Marking Guidelines

For each of the items listed below full marks are awarded if the feature is satisfied, otherwise none (0 marks).

Table 1:

Action	Marks
Chosen instruction is explained	2
Chosen multiply/divide routine is explained	3
Chosen multiply/divide routine is coded for the one instruction CPU	10
CPU data paths are represented diagrammatically with appropriate labels	10
Controller requirements for orchestrating the data path components	10
Total Marks	35

2 Solution

2.1 subneg

The subneg instruction, also called SBN, is defined similarly to subleq

Algorithm 1 subneg a, b, c

Require: c should be a valid jump label
  $M[b] \leftarrow M[b] - M[a]$
  if $M[b] < 0$ **then**
    goto c
  end if

2.2 Multiplication routine

Multiplication can be realized using repetitive addition.

- Say the integers are in registers a and b.
- One of them, say, b can be used as a counter. (Negate both a and b if $b < 0$).
- Increment a zeroed out register c by a, exactly b times.

Algorithm 2 Multiplication routine using repetitive addition

Require: a and b contain integers to be multiplied. ▷ c will contain $a \times b$
  if $b < 0$ **then**
    $b \leftarrow -b$
    $a \leftarrow -a$
  end if
  $c \leftarrow 0$
  while $b > 0$ **do**
    $c \leftarrow c + a$
    $b \leftarrow b - 1$
  end while

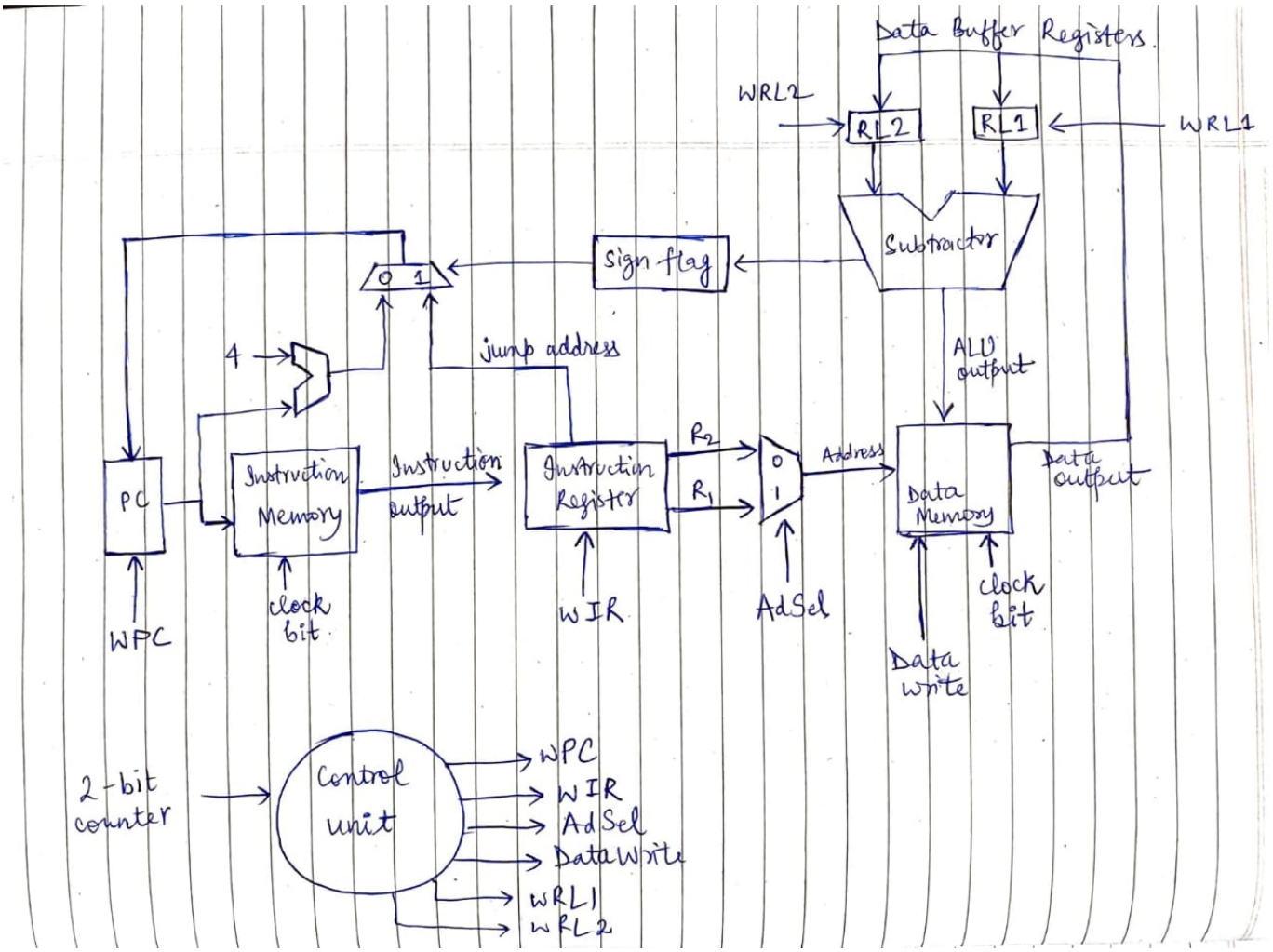
Algorithm 3 Multiplication routine using subneg instruction

Require: $Mem[k]$ should contain value 1

```
    subneg z z                  # z <- 0
    subneg z b negate_b         # check for b < 0
    subneg k z start            # unconditional jump to start

negate_b:
    subneg b z                  # z <- (-b)
    subneg b b                  # b <- 0
    subneg y y                  # y <- 0
    subneg z y                  # y <- b
    subneg y b                  # b <- (-b)
    subneg z z                  # z <- 0

negate_a:
    subneg a z                  # z <- (-a)
    subneg a a                  # a <- 0
    subneg y y                  # y <- 0
    subneg z y                  # y <- a
    subneg y a                  # a <- (-a)

start:
    subneg z z                  # z <- 0
    subneg y y                  # y <- 0
    subneg c c                  # c <- 0

mult:
    subneg a y                  # y <- (-a)
    subneg y c                  # c <- c + a
    subneg k b exit             # b <- b - 1
    subneg k z mult             # unconditional jump to mult

exit:
    subneg a c                  # c <- c - a
```

▷ Mem[c] contains the value Mem[a] × Mem[b]

2.3 Datapath components



- Instruction Format: $[R_1 \mid R_2 \mid jmpAddr]$
- PC is incremented by 4 or by the $jmpAddr$ using a MUX
- ALU subtractor is used to output difference of two values, at the same time controlling the PC using $signFlag$ which is 1 when difference is negative
- There are 6 control signals namely-
  - wPC: write to Program Counter
  - wIR: write to Instruction Register
  - $AdSel$: Instruction register selection line
  - $DataWrite$: write to data memory
  - wRL_1: write to register data buffer RL_1
  - wRL_2: write to register data buffer RL_2

2.4 Controller specifications

The Control Unit has a 2-bit counter has input which controls all the signals. It serves as the state machine to output which control line to use.
The following table describes control values at different values of 2-bit cnt

Table 2:

2-bit cnt	wIR	wPC	wRL_1	wRL_2	DataWrite	AdSel
00	1	0	0	0	0	0
01	0	0	0	1	0	0
10	0	0	1	0	0	1
11	0	1	0	0	1	0

- 2-bit cnt = 00
  - This is the fetch stage
  - Instruction is loaded
- 2-bit cnt = 01
  - Data from R_2 is stored in RL_2
- 2-bit cnt = 10
  - Data from R_1 is stored in RL_1 using $AdSel$ line
- 2-bit cnt = 11
  - Data computed in ALU
  - New output written to Data memory
  - PC incremented accordingly