COMP 30080: Processor Design

2013/2014

Assignments

Dr Chris Bleakley

UCD School of Computer Science and Informatics,
University College Dublin,
Belfield, Dublin 4.

Introduction

There are five assignments. Assignments should be submitted using Moodle via the 'Assignment submission' links. The assignment deadlines are provided in Moodle. Roughly speaking, one assignment is to be submitted every two weeks of term. Moodle will not allow submission after the deadline. Late submissions should be uploaded using the 'Late assignment submission' links. Late submissions incur late submission penalties according to UCD policy unless a medical certificate is submitted to the lecturer. There is a link to the policy document in Moodle.

The last week of term is Demo Week. During Demo Week, students are asked to demonstrate the solutions and simulations that they have developed. The demonstrator will review the demonstration and will ask a number of questions to determine that the student is fully conversant with the material. If the student is not conversant with the work, the student will be asked to re-do the assignments under examination conditions. During Demo Week students may not seek assistance from the demonstrators. Note, only material submitted before the assignment deadlines may be marked during Demo Week.

All students should print off, read, sign and return the UCD submission form to the demonstrators prior to submission of the first assignment. Please use a single form to cover assignments 1-5. A copy of the form is on Moodle.

The assignments do not have to be done during lab times. However, demonstrators are only available during lab times. A record of attendance is taken during lab times. Marks will be available in Moodle.

Some of the assignments use the MARS simulator for simulation of MIPS assembly language programs. It is free and can be downloaded from the link given in Moodle. Read the tutorials available on the MARS web site before installing and using the simulator.

Some of the assignments use the Logisim simulator. It is free and can be downloaded from the link given in Moodle. Read the tutorials available on the web site before installing and using the simulator.

For all assignments:

- Submit the assembly programs that you design as .asm files. Use your student number and question number as file names, e.g. 12345678_ass1_q1a.asm
- Submit the circuits you design as .circ files. Use your student number and question number as file names, e.g. 12345678_ass1_q1a.circ
- Make sure that your name and student number are visible in the circuit once the file is opened.
- For each assignment, submit an accompanying report (.doc or .pdf) which includes screenshots of simulation execution results and the workings for circuit design and the answers for written work questions. Use your student number and assignment number as the file name, e.g. 12345678_ass1.doc
- Make sure that your name and student number are visible in the assembly files, circuits, and report.
- If necessary, scan any handwritten work into a file for submission. Submit the graphics file (.gif or similar) or include the scan in the report.
- Submission is via Moodle. Moodle requires a single file that should be a .zip of all individual files. Use your student number and assignment number as the file name, e.g. 12345678_ass1.zip

PLEASE NOTE, THE REPORTS MUST BE IN WORD OR PDF FORMAT, THE ASSEMBLY PROGRAMS MUST BE IN ASM (TEXT) FORMAT AND THE SUBMISSION MUST BE IN ZIP FORMAT. OTHER FORMATS OR CORRUPT FILES WILL BE TREATED AS LATE SUBMISSIONS.

Assignment 1

Q1. Run the following program in MARS.

```
          .data                         # data goes in data segment
D_in:     .word    2,3,4                # data stored in words
D_out:    .word    5,6,7
          .text                         # code goes in text segment
          .globl   main                 # must be global symbol
main:     la       $t0, D_in            # load address pseudo-instruction
          la       $t1, D_out
          lw       $t2, 8($t0)
          sw       $t2, 0($t1)
          lw       $t2, 4($t0)
          sw       $t2, 4($t1)
          lw       $t2, 0($t0)
          sw       $t2, 8($t1)
          #
          li       $v0, 10              # system call for exit
          syscall                       # Exit!
```

What does the program do?

Change the program so that D_out is equal D_in plus 2, i.e. 2 is added to every element in the array. You don't need to use loops to get full marks.

[Learning outcome: syntax]

Q2. Write a program in MIPS assembly language that will calculate the squares of the numbers 1 to 10 using addition ONLY. The program should use a loop.

Test the program.

[Learning outcome: conditionals]

Q3. Write a program in MIPS assembly language that will determine if two number lists match or not. The program should take the labels of the strings and the length of the strings as input. The program should store a Boolean result (True=match, False=no match) to memory. The program should use loops.

For example, the inputs 1,2,3,4,5 and 1,2,3,4,5 and 5 should give the output True. While, the inputs 0,1,2,3,4 and 1,2,3,4,5 and 5 should give the output False.

Test the program.

[Learning outcome: loops]

Assignment is marked out of 15. Each question is marked out of 5.

0=nothing done; 1=parts done; 2=assembled but not working; 3=almost working; 4=working; 5=working and elegant

Assignment 2

Morse Code was used to send and receive telegraph messages before the invention of the telephone. It was also used for ship radio communications. The coastguard monitored radio frequencies for the Morse SOS distress message until 1999. Morse code represents each character in a message using a sequence of short and long pulses. When written down, the short pulses are represented by dots (.) and the long pulses are represented by dashe[1]s (-). So, SOS is sent as … --- … in Morse code.

1. Implement a MIPS assembly language subroutine called atom that will convert an ASCII upper case letter to its equivalent in Morse code. The Morse representation should be encoded as a string with ASCII dots and dashes and be terminated with a space character. The subroutine input should consist of the ASCII character to be encoded. The subroutine output should consist of a pointer to the Morse representation in memory. For example, input “A” should give output “.- ”.

Hint: The solution will use a hash table indexed by the position of the character in the alphabet. The table should allow for the length of the longest Morse letter: 4 symbols.

Hint: The converter below may be useful for populating the database.

International Morse code can be found at http://en.wikipedia.org/wiki/Morse_code

ASCII code can be found at http://en.wikipedia.org/wiki/ASCII

Useful converter http://www.livephysics.com/ptools/morse-code.php

2. Write a MIPS program called mc that reads in an ASCII string and writes out a string with the message encoded in Morse. The Morse representation should be encoded as a string with ASCII dots and dashes and be terminated with a newline character \n. The program input should consist of the ASCII character string to be encoded held in memory. The program output should consist of the Morse string held in memory. For example, input “CHRIS” should give output “-.-. …. .-. .. … \n”. Test the program using your name.

3. Write a MIPS assembly language procedure called mtoa that will convert the Morse code for a single letter to text. The Morse representation should be encoded as a string with ASCII dots and dashes and be terminated with a space character. The subroutine input should consist of a pointer to the Morse representation in memory. The subroutine output should consist of the ASCII character that was encoded. For example, input “.-” should give output “A”.

Hint: The simplest method is to search the database set up in question 1. Also, see your solution to Assignment 1, Question 3.

Assignment is marked out of 15. Each question is marked out of 5.

0=nothing done; 1=parts done; 2=assembled but not working; 3=almost working; 4=working; 5=working and elegant

[1]

Assignment 3

1. Write a procedure to display a 8x8 prime array, i.e. an 8x8 number grid of the numbers 1 to 64 with the squares associated with the prime numbers colored in. Each square on the grid should be 8x8 pixels.

The program should use the Bitmap Display tool within MARS should be used. Read the Bitmap Display help before starting.

Test the procedure.

[Learning outcome: bit mapped IO]

2. Write a program that will calculate the prime numbers between 1 and 64 using the Sieve of Eratosthenes method.

See http://en.wikipedia.org/wiki/Sieve_of_Eratosthenes

Test the program.

[Learning outcome: algorithmic programming]

3. Write a program that will animate the Sieve of Eratosthenes method by displaying an full grid and deleting the non-prime squares as the algorithm progresses.

The program should be based on your answers to Q1 and Q2.

Test the program.

[Learning outcome: more practice]

Assignment is marked out of 15. Each question is marked out of 5.

0=nothing done; 1=parts done; 2=assembled but not working; 3=almost working; 4=working; 5=working and elegant

miniMIPS

The goal of the project is to build a miniMIPS processor and to run software on it.

The miniMIPS will be implemented using the Logisim simulator. The software will be assembled by hand.

The miniMIPS:

- 4-bit processor
- 4 x 4-bit registers
- 4-bit Data RAM
- 8-bit Program RAM
- 4-bit ALU

Register Set
    $zero, $t0, $t1, $t2

Instruction Set Architecture

```
lw    rt, offset(rs)        # load word from M[rs+offset] to R[rt]
sw    rt, offset(rs)        # store word from R[rt] to M[rs+offset]
addi rt, rs, const          # add immediate R[rt] = R[rs] + const
add  rd, rs, rt             # add R[rd]=R[rs]+R[rt]
```

Instruction Formats
    {opcode; rs; rt; rd/offset/const}
    opcode = 2 bits; rs = 2 bits; rt = 2 bits; rd/offset/const = 2 bits

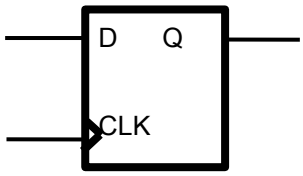
Opcodes
    lw   : opcode = 00
    sw   : opcode = 01
    addi : opcode = 10
    add : opcode = 11

Organisation
    Single cycle MIPS architecture

To do the project you will need to organize into teams of THREE. You will be provided with the Register File circuit described in the next section which you MUST NOT CHANGE. One person in the team should do assignment 4A, one 4B, and one 4C. For Assignment 5, the team should integrate all of the circuits in order to build a complete processor. The team should then test the processor and write and run the program described in Assignment 5. The processor should be demonstrated in Demo Week.

Register File

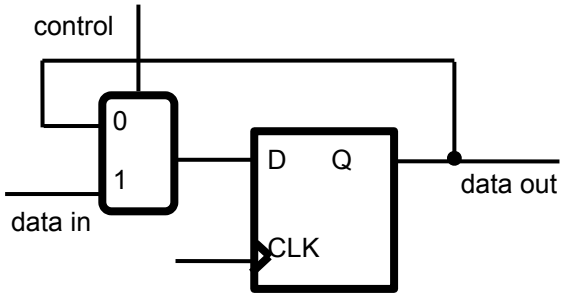
This was an assignment in previous years. This year you will be given a Register File circuit that you MUST NOT CHANGE. The instructions are provided here so that you can understand how it was built.



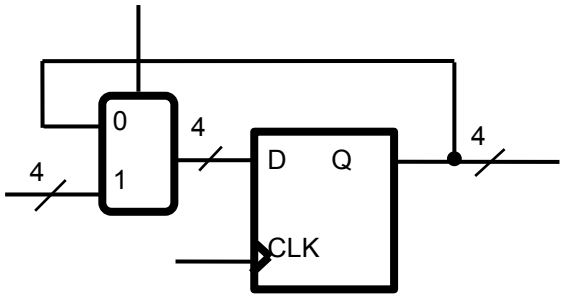
1. Implement a circuit that loads a 1-bit data input signal into a flip-flop on the positive edge of the clock. Connect the data input and clock signals to switches. Connect the flip-flop output to a LED. Demonstrate the working circuit. Show that the output only changes on the positive edge of the clock.

Hint: Use the Data Latch flip-flop type. Check the Edge Triggered box. Qbar can be left unconnected.

2. Extend the circuit developed in question 1, to allow the update of the register to be controlled by another input. The control signal from a switch should be input to the select line of a multiplexer, the output of which should be connected to the input of the flip-flop. The existing data input from the switch should be connected to the 1 input from the multiplier. The multiplier 0 input should be connected to the output of the flip-flop. Demonstrate the circuit to show that the flip-flop will load the data input on the positive edge of the clock when the control signal is 1. When the control signal is 0 the flip-flop output should not change.



The multiplexers require that the Ebar inputs are connected to 0 (ground).



3. Extend the circuit developed in question 1 to implement a 4 bit register with update control. Connect the four bit data input to switches and the four bit data output to LEDs. Test the behaviour of the circuit.

4. Implement a version of the Register File shown in the lecture notes. Implement the Register File with four 4-bit registers, 1 write port and 2 read ports. Use switches for inputs and LEDs for outputs.

Assignment 4A ALU

1. Implement a circuit that has two switch inputs and has two LED outputs. One LED output should be the logic AND of the inputs. The other LED output should be the OR of the inputs. Simulate the circuit to check that it works.

2. Implement a 1-bit full adder. The circuit is shown on the lecture notes. Connect switches to the inputs and LEDs to the outputs. Test that the full adder works. By copying and pasting the 1-bit adder, implement a 4-bit adder. Connect switches to the inputs and LEDs to the outputs. Test that the 4-bit ripple carry adder works.

3. Implement a 4-bit two's-complement ALU which can perform addition or subtraction (a more complicated version is shown in the lecture notes). The ALU should have a 2-bit control input, two 4-bit data inputs and a 4-bit data output. Use switches for inputs and LEDs for outputs.

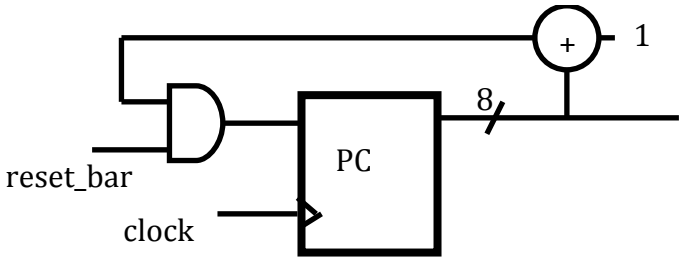
Test the ALU but adding an subtracting the first 2 digits and last 2 digits of your student number.

Assignment is marked out of 15. Each question is marked out of 5.

0=nothing done; 1=parts done; 2=assembled but not working; 3=almost working; 4=working; 5=working and elegant

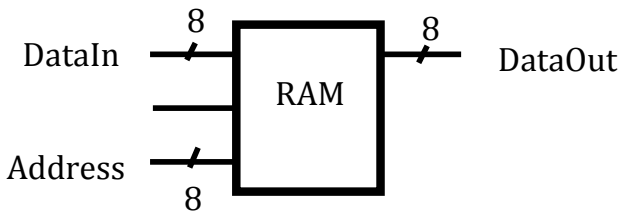
Assignment 4B Program Counter & Program Memory

1. Using the built-in 8-bit adder, build a circuit that implements a 8-bit Program Counter with reset and incrementer. Connect the clock and reset_bar inputs to switches. Connect the Program Counter outputs to LEDs. Test the circuit.



Hint: The signal reset_bar is used to reset the PC to 0.

2. Implement a circuit which allows data to be read and written from a Program RAM (random access



memory). Use a RAM with 8-bit data and 8-bit address lines. Connect the Write control input to a switch. Connect the data output to LEDs. Demonstrate the circuit by writing your student number to the RAM and reading it back. Store each digit in a separate memory location. For example store the student number "08021546" as:

Address	Data
0	0
1	8
2	0
3	2
4	1
5	5
6	4
7	6

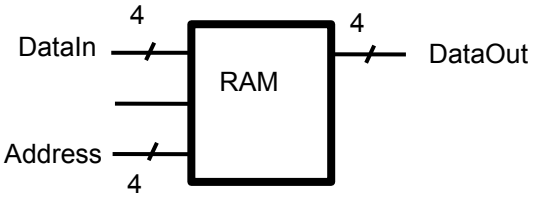
3. Connect the output of the Program Counter registers to the input of the RAM. Load a sequence of data values into the RAM and test that the data values are read out of the RAM in sequence when clocking the Program Counter.

Assignment is marked out of 15. Each question marked out of 5.

0=nothing done; 1=parts done; 2=assembled but not working; 3=almost working; 4=working; 5=working and elegant.

Assignment 4C Data Memory and Control Unit

1. Implement a circuit that allows data to be read and written from a Data RAM (random access memory). Use a RAM with 4-bit data and 4-bit address lines but only connect the 4 Least Significant data bits and 4 Least Significant address bits. This gives a 16 word, 4-bit wide RAM. Connect the Write control input to a switch. Connect the data output to LEDs.



Demonstrate the circuit by writing your student number to the RAM and reading it back. Store each digital in a separate memory location. For example store the student number "08021546" as:

Address	Data
0	0
1	8
2	0
3	2
4	1
5	5
6	4
7	6

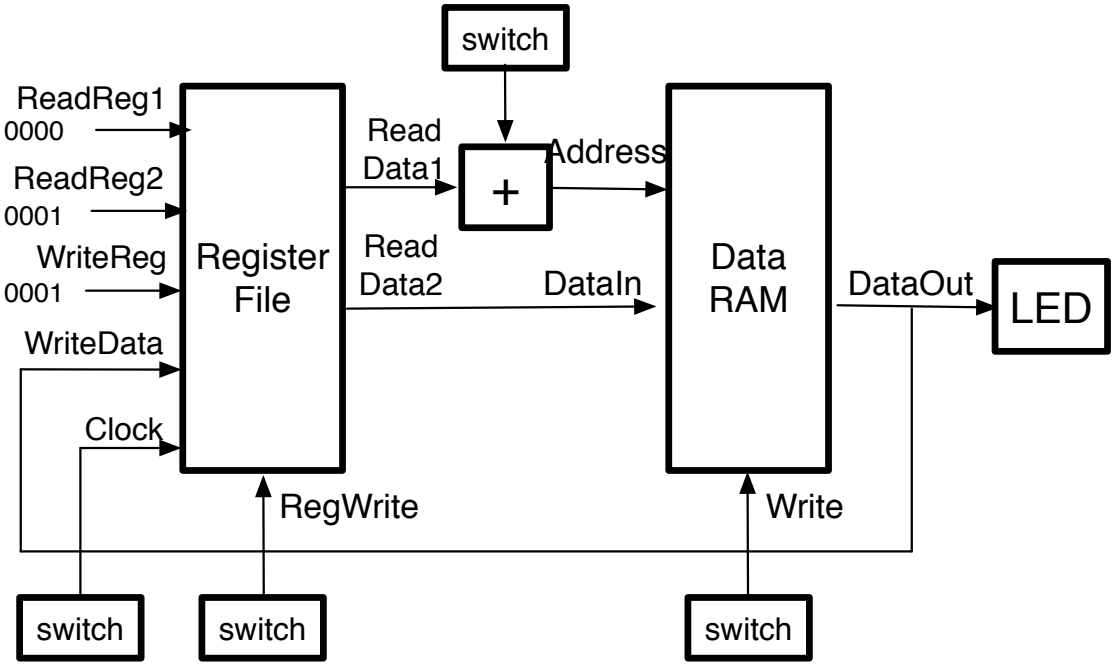
2. Use the built-in 4-bit adder to implement offset addressing for the Data RAM. Test the circuit.

3. Use the Data RAM (see Q1), the 4-bit adder (see Q2) and the Register File (see page 7) to build and test a circuit which will shift the contents of memory up by one address. For example, if the initial contents of RAM are "08021546" then the final contents should be “00802154".

The circuit is illustrated on the next page. In the first clock cycle, the contents of an address should be read from RAM and written to the Register File. In the second clock cycle, the data should be read from the Register File and written to the next location in RAM. This two-step process should repeated for all memory locations. Register $zero is used to store the base address for reading the memory. The offset should be input via switches. Register $1 is used to temporarily store the data.

The LED can be used to check the contents of memory before and after the data is moved.

To make sure that you understand the circuit, draw a timing diagram to illustrate the circuits operation before starting implementation.



The assignment is marked out of 15. Each question is marked out of 5.

0=nothing done; 1=parts done; 2=assembled but not working; 3=almost working; 4=working; 5=working and elegant.

Assignment 5 Complete Processor

1. Write down the true table for the control circuit of the processor. The control circuit should take machine instructions as input and should output the necessary control signals to the Register File, ALU and Data RAM.
2. Design a circuit which will implement the control circuit for the processor. Implement and test the circuit.
3. Integrate the functional units developed in the previous assignments and Q1 into a single processor.
4. Write an assembly program for the miniMIPS processor that will calculate and store the first 8 Triangular numbers in the data memory. The sequence is generated by starting a counter at 1 and incrementing it for every row. The triangular numbers are generated by accumulating the counter.

See http://www.mathsisfun.com/algebra/triangular-numbers.html

5. Manually assemble the program. Load the machine code into Program RAM. Load a string and key into Data RAM. Execute the program by toggling the clock. Check the final results in Data RAM.

Implementation may require some adjustments to the ISA or processor organization.

Assignment is marked out of 20. Each question is marked out of 4. All students in a group get the same mark, UNLESS all students in the team agree that some marks should be transferred from one to the other.

Demo Week

Demonstration of assignments 1,2,3.

Demonstration of the processor

Demonstration of software running on the processor.

0=no demo; 1=some aspects shown; 2=all aspects shown, unable to answer questions; 3=all aspects shown, able to answer simple questions; 4=all aspects shown, able to answer complex questions; 5=all aspects shown, thorough understanding.

Marked out of 25.