

HOMEWORK I

DUE : February 8, 2012

READ :

- Chapter 1 (Except 1.4)
- Related sections of Chapter 2
- Related sections of Chapter 3
- Related portions of Appendix B
- Related portions of Appendix E

ASSIGNMENT : There are seven questions four of which are from chapters I and II of the text-book.

Solve all homework and exam problems as shown in class and past exam solutions.

I) Solve Problems 1.1.1, 1.1.3, 1.1.6, 1.1.8, 1.1.10, 1.1.11, 1.1.13, 1.1.14, 1.1.16, 1.1.17, 1.1.18, 1.1.19, 1.1.20, 1.1.21, 1.1.22, 1.1.23, 1.1.24 and 1.1.25.

To answer each question, write down the word or phrase. Do **not** give the number (1, 2, 3,...) it corresponds to.

II) Solve Problem 2.7.1 part **(b)**, 2.7.2 part **(a)** and 2.7.3 part **(b)**.

Show the work during the conversions as shown in class.

III) Solve Problem 2.8.1 for only part **(a)**.

First, convert the hex digits to bit patterns and then perform additions and subtractions.

IV) Solve Problem 2.10.1 for only part **(b)**.

Show the instruction in the mnemonic notation. Assume that the instruction is in memory location **400000**. Show the work during the conversions as shown in class.

V) Show the minimal sequence of **mnemonic machine language** instructions for this C-like statement :

$$a = b + 100 ;$$

In this question, you are asked to convert a high-level language statement to a set of mnemonic machine language instructions. We know that a compiler converts a high-level language program to a machine language program. So what you will do is exactly the same as what a compiler does, with one exception : you will translate to a mnemonic machine language program, not to a machine language program.

Note the following :

- The set of instructions you will write starts at 400000.
- Variables “a” and “b” are already loaded to registers R8 and R9, respectively.
- Number 100 in the question is a decimal number. But, the **hexadecimal** notation is the default case in our mnemonic machine language programs.

VI) Consider the following C language statement :

$$b = 25 | a$$

Write the corresponding minimal sequence of **mnemonic machine language** program. Assume that the value of “a” is in R8 and the value of “b” should be stored in R9

In order to show clearly that the new code is correct, show the table of execution of instructions with used register and memory location values until the code completes.

Note the following :

- The piece of mnemonic machine language program you will write starts at 400000.
- Assume that R8 has $(6)_{10}$.
- Number 25 in the C-like statement in the question is in decimal. But, the **hexadecimal** notation is the default case in our mnemonic machine language programs.

VII) Implement the following **two** pseudoinstructions in **mnemonic machine language** :

→ MOVE \$t1, \$t2 # \$t1 = \$t2

This pseudo instruction is similar to the one in Problem 2.30.1 part (a) in the textbook

→ CLEAR \$t0 # \$t0 = 0

You will implement the **pseudoinstructions** in terms of (by using) actual EMY instructions.

For each pseudo instruction, you write a mnemonic machine language program that implements the architectural operations required by the pseudo instruction. Note the following :

→ Assume that each mnemonic machine language program you write starts at memory location 400400. Each program looks like as follows :

```
400400 mnemonic rd/rt/rs/Imm/... # Comments are helpful
400404 mnemonic rd/rt/rs/Imm/... # Place the second instruction if necessary
400408 --- --- # Place the third instruction if necessary
```

RELEVANT QUESTIONS AND ANSWERS

Q1) The EMY machine language instruction set does **not** have the following instruction which is shown in the mnemonic notation :

400C00 ADDR M R8, 500(R9) # R8 ← R8 + M[R9 + 500⁺]

i) Implement the instruction : **ADDR M R8, 500(R9)**

by using a few actual EMY instructions in the mnemonic notation. Your piece of code starts at 400C00. Use software conventions discussed in class. Add comments to your code.

ii) Assume that this instruction is added to the EMY instruction set. **Describe** its syntax, semantics, format, etc.

If there is a **new** addressing mode that is **not** discussed in class, indicate so. What does “ADDRM” stand for ?

A1) i) The given instruction is the following :

400C00 ADDR M R8, 500(R9) # R8 ← R8 + M[R9 + 500⁺]

We see that we add a register and a memory operand and store the result in the same register argument. We can implement it by using *two* instructions :

```
400C00 LW R10, 500(R9) # Load the memory operand to R10
400C04 ADD R8, R8, R10 # Add the register and the memory operand
```

ii) The **syntax** of the new instruction : **ADDRM Rt, Disp(Rs)**

The **semantics** of the new instruction : **Rt ← R_t + M[R_s + Disp⁺]**

The format is the **I format** :

6	5	5	16
opcode	Rs	Rt	Displ

Three arguments are used by the instruction : There are **two** register arguments and **a** memory argument.

Rt is a source and destination register. It is explicitly specified by the instruction. One register argument is therefore using the Register addressing mode. The other register argument which is the same register is implied. Therefore, it is using the Implied addressing mode. The third argument is an operand which is a memory location whose address is the sum of a register and a Displacement. Therefore, the 2-byte Signed-Displacement addressing mode is used.

Two memory accesses are made for the instruction. One is to fetch the instruction and the other one is to read a data element.

ADDRM stands for **Add Register Memory**.

EMY is **not** a L/S architecture anymore, since ADDRDM accesses the **memory** for data. The ADDRDM is a CISC instruction.

Q2) A CISC architecture has a powerful complex instruction that stores a constant (a data element) in a memory location MVMC.

The syntax and semantics (the architectural operation) of this instruction, called MVMC (**MoVe to Memory a Constant**) are shown below :

MVMC Rs, Rt, constant # M[Rs + Rt] ← constant⁺

The EMY architecture does not have the MVMC instruction. So, we implement (synthesize) it by using the following piece of mnemonic machine language program which starts at 400200 :

```
400200  ADDI  R8, R0, 5
400204  ADD   R9, R10, R11    # Assume that R10 has 10000350 and R11 has 250
400208  SW    R8, 0(R9)
```

Show the correspondence between the MVMC instruction and this piece of program. For example, which register is Rs, which one is Rt, and so on.

A2) This piece of program corresponds to **MVMC R10, R11, 5** where

- Rs is R10
- Rt is R11
- The constant is a signed number with a value of 5 and a length of 16 bits
- The memory location the constant stored after its sign extension is 100005A0

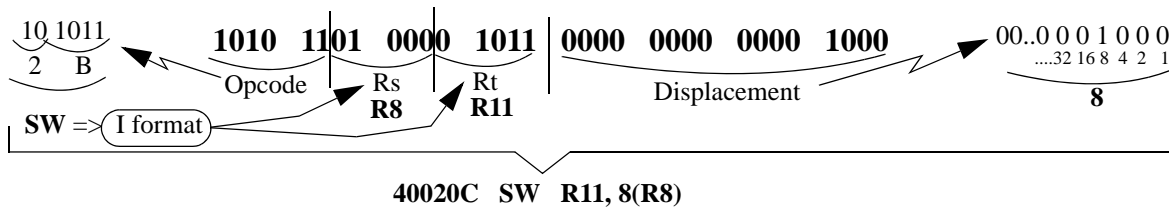
The MVMC instruction can be a **pseudoinstruction** for the MIPS and its conversion to actual MIPS instructions by a MIPS assembler would result in the 3-instruction code above.

Q3) Consider the following memory location with a sequence of bits :

40020C 1010 1101 0000 1011 0000 0000 0000 1000

Determine what the sequence of bits **exactly** represents. Note that this memory location is used for an application for which the MIPS software conventions are followed.

A3) Since the sequence of bits is in 40020C, this must be an instruction according to MIPS software conventions :



Q4) The EMY machine language instruction set does not have the following instruction which is shown in the MIPS assembly notation :

BP \$t7, Label # If \$t7 ≥ 0 then go to Label

Implement the instruction by using EMY instructions in the mnemonic notation. Add a comment to each instruction in your code. Explain what “BP” stands for.

Your piece of code starts at 400250. Memory location “Label” corresponds to 400308. Your piece of code should not contain more than three instructions !

A4) The BP instruction tests a register to see if it is greater than or equal to 0. If yes, it branches to “Label.” Thus, “BP” stands for “Branch if Positive.” That is, if the register specified in the instruction is positive, it is branched. The register specified by the BP instruction is “t7” which corresponds to “R15” in the mnemonic notation.

The corresponding EMY mnemonic machine code has just two instructions as shown below :

```

400250 SLT   R8, R15, R0   # Is R15 less than 0 ?
400254 BEQ   R8, R0, 2C    # If R8 is 0 (R15 is not less than 0), branch 2C locations down.
400258 ---                # Other instructions follow
---
400308 ---
---
```

The “BEQ” in location 400254 has to branch if R15 is positive. It uses the PC-relative addressing mode : an offset in terms of words has to be computed. The distance between 400308 and 400258 is B0 in terms bytes and 2C in terms of words : (B0)/4 = 2C.

The BP instruction can be a pseudoinstruction for the MIPS and its conversion to actual MIPS instructions by a MIPS assembler would result in the 2-instruction code above.

Q5) Consider the following sequence of EMY **mnemonic** machine language instructions :

```

400000      LW      R8, 0(R9)
400004      SLT      R10, R8, R0
400008      BEQ      R10, R0, 1
40000C      SW      R0, 0(R9)
400010      ADDI     R9, R9, 4
400014      ADDI     R11, R11, (-1)10
400018      BNE      R11, R0, (-7)10
    
```

i) Obtain the table that shows the **values** of registers and memory locations used by the above piece of EMY code. To do that, continue with the table below :

Instruction	PC	R8	R9	R10	R11	M[10000000]
Initial	400000	?	10000000	?	1	$(-5)_{10}$
LW R8, 0(R9)	400004	$(-5)_{10}$	NS	NS	NS	NS
.....	Continue

ii) Determine what this piece of code does. That is, what is its **purpose** ?

A5)

i) The table that shows the values of registers and memory locations used by the code is as follows :

Instruction	PC	R8	R9	R10	R11	M[10000000]
Initial	400000	?	10000000	?	1	$(-5)_{10}$
LW R8, 0(R9)	400004	$(-5)_{10}$	NS	NS	NS	NS
SLT R10, R8, R0	400008	NS	NS	1	NS	NS
BEQ R10, R0, 1	40000C	NS	NS	NS	NS	NS
SW R0, 0(R9)	400010	NS	NS	NS	NS	0
ADDI R9, R9, 4	400014	NS	10000004	NS	NS	NS
ADDI R11, R11, (-1)₁₀	400018	NS	NS	NS	0	NS
BNE R11, R0, (-7)₁₀	40001C	NS	NS	NS	NS	NS

ii) This piece of code **zeros negative** elements of a one-dimensional **array** in the memory. The *positive* elements of the array are **not** changed. For the given code, the array has only one element which is negative. Therefore, it is replaced with zero.

Q6) Consider the following piece of EMY **mnemonic** machine language program :

```

400A50      LW      R8, 0(R9)           # R9 initially has 10000000
400A54      LW      R10, 500(R9)
400A58      SLT     R11, R10, R0
400A5C      BEQ     R11, R0, 1
400A60      SW      R8, 500(R9)
400A64      ADDI    R9, R9, 4
400A68      ADDI    R12, R12, (-1)10    # R12 initially has 2
400A6C      BNE     R12, R0, (-8)10
-----
10000000    E
10000004    A
-----
10000500    5
10000504    (-1)10

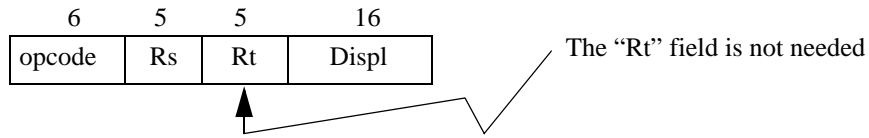
```

- i) Obtain** a table that shows the values of registers and memory locations used by the above piece of EMY code as shown in class. Also **show** the number of memory accesses made for each instruction.
- ii) Determine** what this piece of code does. That is, what is its purpose in a few sentences ?
- iii) Invent** a new EMY instruction that combines the SLT and BEQ instructions in order to reduce the size of the code. **Indicate** the syntax, semantics and format of this new instruction.

A6) i) The table showing the values of registers and memory locations used by the code is as follows :

Instruction	PC	R8	R9	R10	R11	R12	M[10000000]	M[10000004]	M[10000500]	M[10000504]	Mem. Acc.
Initial	400A50	?	10000000	?	?	2	E	A	5	(-1) ₁₀	-----
LW R8, 0(R9)	400A54	E	NS	NS	NS	NS	NS	NS	NS	NS	2
LW R10, 500(R9)	400A58	NS	NS	5	NS	NS	NS	NS	NS	NS	2
SLT R11, R10, R0	400A5C	NS	NS	NS	0	NS	NS	NS	NS	NS	1
BEQ R11, R0, 1	400A64	NS	NS	NS	NS	NS	NS	NS	NS	NS	1
ADDI R9, R9, 4	400A68	NS	10000004	NS	NS	NS	NS	NS	NS	NS	1
ADDI R12, R12, (-1) ₁₀	400A6C	NS	NS	NS	NS	1	NS	NS	NS	NS	1
BNE R12, R0, (-8) ₁₀	400A50	NS	NS	NS	NS	NS	NS	NS	NS	NS	1
LW R8, 0(R9)	400A54	A	NS	NS	NS	NS	NS	NS	NS	NS	2
LW R10, 500(R9)	400A58	NS	NS	(-1) ₁₀	NS	NS	NS	NS	NS	NS	2
SLT R11, R10, R0	400A5C	NS	NS	NS	1	NS	NS	NS	NS	NS	1
BEQ R11, R0, 1	400A60	NS	NS	NS	NS	NS	NS	NS	NS	NS	1
SW R8, 500(R9)	400A64	NS	NS	NS	NS	NS	NS	NS	NS	A	2
ADDI R9, R9, 4	400A68	NS	10000008	NS	NS	NS	NS	NS	NS	NS	1
ADDI R12, R12, (-1) ₁₀	400A6C	NS	NS	NS	NS	0	NS	NS	NS	NS	1
BNE R12, R0, (-8) ₁₀	400A70	NS	NS	NS	NS	NS	NS	NS	NS	NS	1

Since we need a displacement, we need to use the **I-format** :



The instruction has **two** architectural operations : The first one increments a memory location and the second adds 4 to a register.

For the **first** architectural operation : There are three arguments. For the destination argument and the first source argument the 2-byte signed displacement addressing mode is used. For the second source argument, the implied addressing mode is used since a "1" is added and "1" is implied by the instruction

For the **second** architectural operation : There are three arguments. For all the three arguments, the implied addressing mode is use. The destination argument and first source argument are implied to be register "Rs" and the second source argument is implied to be "4" by the instruction.

```
ii)   100   INCMAI   0(R9)++
       104   ADDI    R10, R10, (-1)10
       108   BNE    R10, R0, (-3)10
```

These three instructions are executed R10 times. Then, the number of instructions executed is 3 x R10.

For the original program, the number of instructions executed is 6 x R10 ! The number of instructions executed is decreased by half ! Note that the **EMY computer is not a L/S (RISC) computer anymore** since an A/L instruction, INCM, accesses the memory for data.

Q8) Consider the following piece of EMY **mnemonic** machine language instructions :

```
400000   ADDI   R10, R0, 0
400004   ADD    R10, R8, R10
400008   ADDI   R9, R9, (-1)10
40000C   BNE   R9, R0, (-3)10
```

i) Obtain the table that shows the **values** of registers and memory locations used by the above piece of EMY code. To do that, continue with the table below as shown in class :

Instruction	PC	R8	R9	R10
Initial	400000	9	2	?
.....	Continue

ii) Determine what this piece of code does. That is, what is its **purpose** in a few sentences ?

A8i) The table showing the values of registers and memory locations used by the code is as follows :

Instruction	PC	R8	R9	R10	Comments
Initial	400000	9	2	?	
ADDI R10, R0, 0	400004	NS	NS	0	Initialize R10 to 0
ADD R10, R8, R10	400008	NS	NS	9	Add R8 to R10
ADDI R9, R9, (-1)₁₀	40000C	NS	1	NS	Decrement the loop counter
BNE R9, R0, (-3)₁₀	400004	NS	NS	NS	Not the end, go back to 400004
ADD R10, R8, R10	400008	NS	NS	(18)₁₀	Add R8 to R10 again
ADDI R9, R9, (-1)₁₀	40000C	NS	0	NS	Decrement the loop counter
BNE R9, R0, (-3)₁₀	400010	NS	NS	NS	It is the end, exit the loop

ii) This piece of code **multiplies** two *positive* numbers by performing successive **additions**. It also works if the multiplicand of the (multiplicand * multiplier) pair is negative. For example, $(-9 * 2)$ would work. The multiplicand must be **at least 1** initially. The loop execution is shown in bold face on the above execution table.

Q9) The EMY computer runs a piece of program and the following values are observed :

PC	R2	R4	R8	R31	M[10004000]	M[10004004]
400300	?	10004000	?	154	51	$(-1)_{10}$
400304	0	NS	NS	NS	NS	NS
400308	NS	NS	51	NS	NS	NS
40030C	NS	NS	52	NS	NS	NS
400310	1	NS	NS	NS	NS	NS
400314	NS	10004004	NS	NS	NS	NS
400304	NS	NS	NS	NS	NS	NS
400308	NS	NS	$(-1)_{10}$	NS	NS	NS
40030C	NS	NS	0	NS	NS	NS
400310	2	NS	NS	NS	NS	NS
400314	NS	10004008	NS	NS	NS	NS
400318	NS	NS	NS	NS	NS	NS
400154	NS	NS	NS	NS	NS	NS

The first row shows the initial values.

- a) Write the corresponding mnemonic machine language program.
 b) Describe what the above program does briefly (in three sentences).

A9) a)

```

400300 ADD    R2, R0, R0          # R2 is initialized to 0
400304 LW     R8, 0(R4)          # R8 is loaded from an array pointed by R4
400308 ADDI   R8, R8, 1          # The array element is incremented by 1
40030C ADDI   R2, R2, 1          # R2 is incremented by 1
400310 ADDI   R4, R4, 4          # The array pointer (R4) is updated to point at the next element
400314 BNE    R8, R0, (-5)10    # If the loaded array element plus 1 is not 0, it branches back
400318 JR     R31                # It returns from the subroutine
  
```

b) This is a **subroutine** that counts the number of elements of an array that starts at R4 (the array pointer). It returns the count plus 1 in R2. The last element of the array is always $(-1)_{10}$. In the question, the array starts at 10004000 and has one element and so R2 contains a 2.

Q10) Write a piece of EMY **mnemonic** machine language code that **negates** the elements of an array.

The array is pointed by R8. The number of elements in the array is specified by R9 which is **not zero** *initially*. The code starts at 400000.

Use software conventions discussed in class. Add comments to your code. If the code contains more than **eight** (8) instructions, it will **not** earn points.

A10) The code is as follows :

```

400000      LW      R10, 0(R8)      # Load from the array
400004      SUB     R10, R0, R10    # Negate the element
400008      SW      R10, 0(R8)     # Store back to the array
40000C      ADDI   R8, R8, 4        # Update the array pointer
400010      ADDI   R9, R9, (-1)10  # Decrement the loop index
400014      BNE    R9, R0, (-6)10  # If not the end, loop back to 400000
  
```

Q11) Consider the following piece of EMY **mnemonic** machine language program :

```

400100      ADDI   R8, R0, 0
400104      LW     R9, 0(R10)      # R10 initially has 10001000
400108      ADD    R8, R8, R9
40010C      ADDI   R10, R10, 4
400110      ADDI   R11, R11, (-1)10 # R11 initially has 2
400114      BNE    R11, R0, (-5)10
400118      SW     R8, 0(R10)
-----
10001000    6
10001004    4
  
```

i) Obtain the table that shows the **values** of registers and memory locations used by the above piece of EMY code as shown in class :

Instruction	PC	R8	R9	R10	R11	M[10001000]	M[10001004]	M[10001008]
Initial	400100	?	?	10001000	2	6	4	?
ADDI R8, R0, R0	400104	0	NS	NS	NS	NS	NS	NS
....	Continue

ii) Determine what this piece of code does. That is, what is its **purpose** in a few sentences ?

A11)

i) The table showing the values of registers and memory locations used by the code is as follows :

Instruction	PC	R8	R9	R10	R11	M[10001000]	M[10001004]	M[10001008]
Initial	400100	?	?	10001000	2	6	4	?
ADDI R8, R0, 0	400104	0	NS	NS	NS	NS	NS	NS
LW R9, 0(R10)	400108	NS	6	NS	NS	NS	NS	NS
ADD R8, R8, R9	40010C	6	NS	NS	NS	NS	NS	NS
ADDI R10, R10, 4	400110	NS	NS	10001004	NS	NS	NS	NS
ADDI R11, R11, (-1)₁₀	400114	NS	NS	NS	1	NS	NS	NS
BNE R11, R0, (-5)₁₀	400104	NS	NS	NS	NS	NS	NS	NS
LW R9, 0(R10)	400108	NS	4	NS	NS	NS	NS	NS
ADD R8, R8, R9	40010C	A	NS	NS	NS	NS	NS	NS
ADDI R10, R10, 4	400110	NS	NS	10001008	NS	NS	NS	NS
ADDI R11, R11, (-1)₁₀	400114	NS	NS	NS	0	NS	NS	NS
BNE R11, R0, (-5)₁₀	400118	NS	NS	NS	NS	NS	NS	NS
SW R8, 0(R10)	40011C	NS	NS	NS	NS	NS	NS	A

ii) This piece of code **adds** elements of an array **one by one** and stores the result in the location right after the array. In the given problem, there are two elements.

Q12) Consider the following piece of EMY mnemonic machine language program :

```

400300    LW      R9, 0(R8)          # R8 initially has 10000000
400304    ANDI    R10, R9, 8000
400308    BEQ     R10, R0, 3
40030C    LUI     R11, FFFF
400310    OR      R9, R11, R9
400314    SW      R9, 0(R8)
400318    ADDI    R8, R8, 4
40031C    ADDI    R12, R12, (-1)10    # R12 initially has 1
400320    BNE     R12, R0, (-9)10
-----
10000000    A5F2

```

a) Obtain a table that shows the values of registers and memory locations used by the above piece of EMY code as shown in class. Also **show** the number of memory accesses made for each instruction. Determine what this piece of code does. That is, what is its **purpose** ?

b) Convert the mnemonic machine language instructions in locations 400304 - 400310 to machine language instructions. That is, **clearly** show the **four** instructions in terms of **1s** and **0s**.

c) Invent a new EMY instruction that implements the instructions in locations 400304 - 400310 above : Indicate **only** the following : Its syntax, semantics, format and the memory accesses made. Then, rewrite the above code starting at 400300 so that the **new** EMY instruction is used.

A12) a) The table showing the values of registers and memory locations used by the code is as follows :

Instruction	PC	R8	R9	R10	R11	R12	M[10000000]	Memory Accesses
Initial	400300	10000000	?	?	?	1	A5F2	-----
LW R9, 0(R8)	400304	NS	A5F2	NS	NS	NS	NS	2 : IF & DR
ANDI R10, R9, 8000	400308	NS	NS	8000	NS	NS	NS	1 : IF
BEQ R10, R0, 3	40030C	NS	NS	NS	NS	NS	NS	1 : IF
LUI R11, FFFF	400310	NS	NS	NS	FFFF0000	NS	NS	1 : IF
OR R9, R11, R9	400314	NS	FFFFA5F2	NS	NS	NS	NS	1 : IF
SW R9, 0(R8)	400018	NS	NS	NS	NS	NS	FFFFA5F2	2 : IF & DW
ADDI R8, R8, 4	40031C	10000004	NS	NS	NS	NS	NS	1 : IF
ADDI R12, R12, (-1) ₁₀	400320	NS	NS	NS	NS	0	NS	1 : IF
BNE R12, R0, (-9) ₁₀	400324	NS	NS	NS	NS	NS	NS	1 : IF

This code sign extends 2-byte data elements stored in memory starting at 10000000.

b) The machine language instructions in locations 400304 - 400310 are as follows :

```

400304    001100 01001 01010 1000 0000 0000 0000
400308    000100 01010 00000 0000 0000 0000 0011
40030C    001111 00000 01011 1111 1111 1111 1111
400310    000000 01011 01001 01001 00000 100101

```

c) The four instructions sign extend a 2-byte data element in a register. This is the **SGNEXT** instruction :

Syntax : SGNEXT Rt, Rs

Semantics : If Rs[15] = 1 then Rt ← (FFFF),(Rs[15:0])

Format, etc. :

- It can use the **I** or **R** formats since only Rs and Rt are needed. If the **I** format is used, DOImm is not used. Otherwise, Rd is not used.
- We make **one** memory access for the new instruction to fetch it

The SGNEXT instruction saves **four** original instructions. The original code is rewritten with the SGNEXT below :

400000	LW	R9, 0(R8)	# Load a data element from the memory
400004	SGNEXT	R9, R9	# Sign extend the data element
400008	SW	R9, 0(R8)	# Store the sign extended number to the memory
40000C	ADDI	R8, R8, 4	# Update memory pointer
400010	ADDI	R12, R12, (-1) ₁₀	# Decrement the loop-end counter
400014	BNE	R12, R0, (-6) ₁₀	# If not the end, go back to 400000

Q13) Consider the following pseudoinstruction in the mnemonic machine language notation :

SWCEQ Rd, Rs, Rt ==> If Rs = Rt then M[Rd] ← R(d+1)

If **Rd** is R8 then **R(d+1)** is R9. Store Word Conditional (SWCEQ) stores to the memory if the condition of the instruction is satisfied. For example :

SWCEQ R8, R10, R11 ==> If R10 = R11 then M[R8] ← R9

a) Implement the instruction

SWCEQ R8, R10, R11

by using a few actual EMY instructions in the mnemonic notation. Your piece of code starts at 400F00. Use software conventions discussed in class. **Add** comments to your code.

b) Assume that this instruction is added to the EMY instruction set. **Indicate** its format, arguments, addressing modes, memory accesses made, etc. If there is a **new** addressing mode that is **not** discussed in class, indicate so.

c) Consider the following piece of EMY mnemonic machine language program :

4000E0	LW	R8, 0(R9)	# R9 has 10000000 initially
4000E4	ADD	R10, R8, R11	# R11 has 7 initially
4000E8	SWCEQ	R9, R10, R12	# R12 has 8 initially
4000EC	ADDI	R9, R9, 4	
4000F0	ADDI	R13, R13, (-1) ₁₀	# R13 has 1 initially
4000F4	BNE	R13, R0, (-6) ₁₀	
----	----		
10000000	1		

Obtain a table that shows the values of registers and memory locations used by the above piece of EMY code as shown in class. Also **show** the number of memory accesses made for each instruction.

A13) a) The implementation of the “SWCEQ R8, R10, R11” is as follows :

```
400F00    BNE    R10, R11, 1    # If R10 is not equal to R11 then skip the SW instruction
400F04    SW     R9, 0(R8)      # Store R9 to the memory pointed by R8
```

b) Format, etc. :

- It uses the **R** format since we need Rd. Shamt is **not** used.
- It has **four** arguments.
 - Rs and Rt are source registers and they are explicitly specified by the instruction : Both arguments use the Register addressing mode.
 - “R(d+1) is a source register and is implied : It uses the Implied addressing mode.
 - The destination is a memory location whose address is contained by a register (Rd) which is explicitly specified. This is the register indirect addressing mode, **not** discussed in class.
- We make at most **two** memory access for the new instruction depending on the condition :
 - One to fetch the instruction
 - If the condition is satisfied, one more memory access to write data

c) The execution table is as follows :

Instruction	PC	R8	R9	R10	R11	R12	R13	M[10000000]	Mem. Acc.
Initial	4000E0	?	10000000	?	7	8	1	1	-----
LW R8, 0(R9)	4000E4	1	NS	NS	NS	NS	NS	NS	2 : IF & DR
ADD R10, R8, R11	4000E8	NS	NS	8	NS	NS	NS	NS	1 : IF
SWCEQ R9, R10, R12	4000EC	NS	NS	NS	NS	NS	NS	8	2 : IF & DW
ADDI R9, R9, 4	4000F0	NS	10000004	NS	NS	NS	NS	NS	1 : IF
ADDI R13, R13, (-1) ₁₀	4000F4	NS	NS	NS	NS	NS	0	NS	1 : IF
BNE R13, R0, (-6) ₁₀	4000F8	NS	NS	NS	NS	NS	NS	NS	1 : IF

Q14) Consider the following piece of EMY mnemonic machine language program :

```
400400    LW     R8, 0(R9)      # R9 initially has 10000000
400404    AND    R8, R8, R10    # R10 initially has 4FFFFFFF
400408    NOR    R8, R8, R8
40040C    SLL    R8, R8, 1
400410    SLT    R11, R8, R0
400414    BEQ    R11, R0, 2
400418    ADDI   R9, R9, 4
40041C    J     100100
400420    SW     R8, 4(R9)
-----
10000000    F0000000
```

a) Obtain a table that shows the values of registers and memory locations used by the above piece of EMY code as shown in class. Also **show** the number of memory accesses made for each instruction.

b) Convert the mnemonic machine language instructions in locations 400404 - 400408 to machine language instructions. That is, **clearly** show the **two** instructions in terms of **1s** and **0s**.

c) **Invent** a new EMY instruction that implements the instructions in locations 400404 - 400408 above : Indicate **only** the following : Its syntax, semantics, format and the memory accesses made. Then, rewrite the above code starting at 400400 so that the **new** EMY instruction is used. **Add** comments to your code.

A14) a) The table showing the values of registers and memory locations used by the code is as follows :

Instruction	PC	R8	R9	R10	R11	M[10000000]	M[10000004]	Memory Accesses
Initial	400400	?	10000000	4FFFFFFF	?	F0000000	?	-----
LW R8, 0(R9)	400404	F0000000	NS	NS	NS	NS	NS	2 : IF & DR
AND R8, R8, R10	400408	40000000	NS	NS	NS	NS	NS	1 : IF
NOR R8, R8, R8	40040C	BFFFFFFF	NS	NS	NS	NS	NS	1 : IF
SLL R8, R8, 1	400410	7FFFFFFE	NS	NS	NS	NS	NS	1 : IF
SLT R11, R8, R0	400414	NS	NS	NS	0	NS	NS	1 : IF
BEQ R11, R0, 2	400420	NS	NS	NS	NS	NS	NS	1 : IF
SW R8, 4(R9)	400424	NS	NS	NS	NS	NS	7FFFFFFE	2 : IF & DW

The logic operations are performed as follows :

AND R8, R8, R10	R8	4FFFFFFF	0100 1111 1111 1111 1111 1111 1111 1111
	R10	F0000000	1111 0000 0000 0000 0000 0000 0000 0000
	R8	40000000	0100 0000 0000 0000 0000 0000 0000 0000
NOR R8, R8, R8	R8	40000000	0100 0000 0000 0000 0000 0000 0000 0000
	R8	40000000	0100 0000 0000 0000 0000 0000 0000 0000
	R8	BFFFFFFF	1011 1111 1111 1111 1111 1111 1111 1111

b) The machine language instructions in locations 400404 - 400408 are as follows :

```
400404    000000 01000 01010 01000 00000 100100
400408    000000 01000 01000 01000 00000 100111
```

c) The NOR instruction NORs the same register which results in the complement of the register. Therefore, we have an AND operation followed by an invert operation. Then, the two instructions perform a NAND operation on two data elements that are in two registers. This is the **NAND** instruction :

Syntax : NAND Rd, Rs, Rt

Semantics : Rd ← (Rs & Rt)

Format, etc. :

- **R** format is used since Rd is needed. Shamt is **not** used.
- We make **one** memory access for the new instruction to fetch it

The NAND instruction saves **one** original instruction. The original code is rewritten with the NAND below :

```
400400    LW      R8, 0(R9)      # Load a data element from the memory
400404    NAND   R8, R8, R10   # NAND registers R8 and R10
400408    SLL    R8, R8, 1      # Shift left R8 by one bit position
40040C    SLT    R11, R8, R0    # Is the result negative ?
400410    BEQ    R11, R0, 2     # If no, skip next two instructions
400414    ADDI   R9, R9, 4      # Otherwise, increment the index register, R9
400418    J      100100        # Then, go back to beginning of the loop, the LW instruction
40041C    SW     R8, 4(R9)     # The end: Store the result in memory following last data location
```