*Advanced Computer Architecutre Lab 2018*

1 – Warm-up questions

* 1. ADD R2 R0 5
  2. SUB R2 R0 5
  3. SUB R2 R2 R3
  4. Regarding “>” and “>=”, it can be implemented by replacing the src0 and src1 numbers in JLT and JLE instructions.
  5. In order to load a 32 bit constant into a register, one can use multiple instructions. For example, if the constant is 0x89ABCDEF and the register is R2:  
     OR R2 R0 0XCDEF

LHI R2 0X89AB

1. – Example program
   1. The program does the first step in an array scan algorithm.

Namely, it makes an addition of each couple of memory content and stores it the first.

Memory contents before the run:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

Memory contents after the run:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 |
| 1 | 3 | 5 | 7 | 9 | 11 | 13 | 7 |

* 1. The inputs are stored in memory addresses 15-22 (including).
  2. The outputs are stored in memory addresses 15-21 (including).
  3. The commented version of the assembly program:

asm\_cmd(ADD, 2, 1, 0, 15); // 0: R2 = 15

asm\_cmd(ADD, 3, 1, 0, 1); // 1: R3 = 1

asm\_cmd(ADD, 4, 1, 0, 8); // 2: R4 = 8

asm\_cmd(JEQ, 0, 3, 4, 11); // 3: if (R3 == R4) goto 11

asm\_cmd(LD, 5, 0, 2, 0); // 4: R5 = MEM[R2]

asm\_cmd(ADD, 2, 2, 1, 1); // 5: R2++

asm\_cmd(LD, 6, 0, 2, 0); // 6: R6 = MEM[R2]

asm\_cmd(ADD, 6, 6, 5, 0); // 7: R6 += R5

asm\_cmd(ST, 0, 6, 2, 0); // 8: MEM[R2] = R6

asm\_cmd(ADD, 3, 3, 1, 1); // 9: R3++

asm\_cmd(JEQ, 0, 0, 0, 3); // 10: goto 3

asm\_cmd(HLT, 0, 0, 0, 0); // 11: HALT

* 1. The program can use immidieates of the numbers 0-7 instead of using the memory content, and only stores the outputs. The new program:

asm\_cmd(ADD, 2, 0, 0, 15); // 0: R2 = 0

asm\_cmd(ADD, 3, 1, 0, 1); // 1: R3 = 1

asm\_cmd(ADD, 4, 1, 0, 8); // 2: R4 = 8

asm\_cmd(JEQ, 0, 3, 4, 11); // 3: if (R3 == R4) goto 11

asm\_cmd(ADD, 5, 0, 2, 0); // 4: R5 = R2

asm\_cmd(ADD, 2, 2, 1, 1); // 5: R2++

asm\_cmd(ADD, 6, 0, 2, 0); // 6: R6 = R2

asm\_cmd(ADD, 6, 6, 5, 0); // 7: R6 += R5

asm\_cmd(ST, 0, 6, 2, 0); // 8: MEM[R2] = R6

asm\_cmd(ADD, 3, 3, 1, 1); // 9: R3++

asm\_cmd(JEQ, 0, 0, 0, 3); // 10: goto 3

asm\_cmd(HLT, 0, 0, 0, 0); // 11: HALT

1. ISS simulator Testing #1

The following is the annotated assembly code of the multiply program.

It uses the standard integer binary multiply algorithm, which involves simple left-shifts and additions. This enables also signed numbers multiplication, written in two’s complement form. Additionally, in contrast to the brute force way of multiplication, this algorithm runtime is constant (always 32 iterations, depending on the register size), and does not depend on the size of the input numbers.

Note that this method enables multiplying numbers sized up to 16 bits, also preventing overflow.

LD R3, R0, R1, 1000 # R3 = Mem[1000] = x

LD R4, R0, R1, 1001 # R4 = Mem[1001] = y

ADD R2, R0, R1, 1 # R2 = mask = 1

ADD R6, R0, R0, 0 # R6 = 0 --> result

ADD R5, R0, R1, 32 # R5 = i = 32

loop: AND R7, R2, R3, 0 # R5 = x & R2

JEQ R0, R0, R7, shift # if R5==0 goto shift

ADD R6, R6, R4, 0 # else, do addition: result += y

shift: LSF R2, R2, R1, 1 # mask << 1

LSF R4, R4, R1, 1 # y << 1

SUB R5, R5, R1, 1 # i--

JLT R0, R0, R5, loop # if i > 0 goto loop

ST R0, R6, R1, 1002 # Mem[1002] = result

HALT

The final assembly does not include labels, they were left here for readability, and were replaced in the code with the appropriate line number.

Mult.bin file contains the code, and the two input numbers in memory:

Mem[1000] = 50 (0x00000032)

Mem[1001] = -100 (0xffffff9c)

Mult\_trace.txt and mult\_sram\_out.txt are attached to this assignment.

Mult\_sram\_out.txt contains the two input numbers in place, and also the output number:

Mem[1002] = -5000 (0x ffffec78)

1. ISS simulator Testing #2

The following is the annotated assembly code of the multiplication table calculation program.

It uses a time efficient algorithm which does not involve using multiplication at all, but additions and memory instructions.

The program starts by initializing the first row of the table each cell at a time.

By the 2nd row, the main loop starts. Each iteration it adds the index of the column (valued from 1 to 10) to the value of the cell on the same column and the previous row, writing in in the current cell.

Note that this algorithm uses a lot of memory transactions, but it’s nevertheless a lot efficient than using multiplication algorithm each time (230~ instructions rather than 20,000 and more).

ADD R4, R0, R1, 10 # R4 = res = 10

initloop: JEQ R0, R4, R0, main

ADD R2, R4, R1, 1999 # address = 2000 + res - 1

ST R0, R4, R2, 0 # mem[address] = res

SUB R4, R4, R1, 1 # res--

JIN R0, R1, R0, initloop

main: ADD R2, R0, R1, 2010

ADD R3, R0, R1, 1 #R3 = row = 1

ADD R4, R0, R0, 0 #R4 = col = 0

rowloop: ADD R7, R0, R1, 10

JEQ R0, R3, R7, finish # if row == 10 finish

colloop: ADD R7, R0, R1, 10

JEQ R0, R4, R7, finishloop # if col == 10

SUB R7, R2, R1, 10

LD R7, R0, R7, 0 # tmp = mem[address - 10]

ADD R7, R7, R4, 0 #

ADD R7, R7, R1, 1 # tmp += col + 1

ST R0, R7, R2, 0 # mem[address] = tmp

ADD R4, R4, R1, 1 # col++

ADD R2, R2, R1, 1

JIN R0, R1, R0, colloop

finishloop:ADD R3, R3, R1, 1 # row++

JIN R0, R1, R0, rowloop

finish: HLT R0, R0, R0, 0

The final assembly does not include labels, they were left here for readability, and were replaced in the code with the appropriate line number.

Mult\_table.bin, Mult\_table\_trace.txt, Mult\_table\_sram\_out.txt – are attached to this assignment.