0512.4490 Advanced Computer Architecture Lab 2013

Lab #1: ASIC hierarchical verification: high level simulator

When designing a new processor, we typically construct simulators of the new processor before designing the architecture details, and before coding it in RTL. This allows us to catch bugs early in the design process, as well as provide a reference for future lower level simulators, and RTL verification.

In this lab we'll model a simple processor by constructing a C high level ISS (instruction set simulator). We'll build the simulator, and test its functionality.

SP (simple processor) specification

SP is a 32 bit processor. It contains 6 registers names r2 to r7, each 32 bits. A single static SRAM of size 65536 words of 32-bit each is used for both program and data. A single 32-bit instruction format is used:

```
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
< 0 > <---opcode---> <-dst--> <-src0-> <-src1-> <---------------- immediate ------------------>
```

dst specifies the destination register r2 .. r7, or r0 in case no register write back is performed.
src0, src1specify the two source registers r2 to r7. 0 in the source field will mean 0, and 1 will mean the immediate field, sign extended.

opcode specifies the instruction operation, and is one of the following:

Arithmetic/Logic operations:

R[dst] = R[src0] operation R[src1], where operation depends on the opcode, and R[0], R[1] have special meaning as described above:

0: ADD: addition: R[dst] = R[src0] + R[src1]
1: SUB: subtraction: R[dst] = R[src0] - R[src1]
2: LSF: left shift: R[dst] = R[src0] << R[src1]
3: RSF: (signed) right shift: R[dst] = R[src0] >> R[src1]
4: AND: bitwise logical and: R[dst] = R[src0] & R[src1]
5: OR: bitwise logical or: R[dst] = R[src0] | R[src1]
6: XOR: bitwise logical xor: R[dst] = R[src0] ^ R[src1]
7: LHI: load the high bits [31:16] of the destination register with immediate[15:0]

Load/Store

8: LD: loads memory contents at address specified by R[src1]
9: ST: writes R[src0] to memory at address R[src1]

Flow Control

16: JLT: jumps to immediate[15:0] if R[src0] < R[src1]
17: JLE: jumps to immediate[15:0] if R[src0] <= R[src1]
18: JEQ: jump to immediate[15:0] if R[src0] == R[src1]
19: JNE: jump to immediate[15:0] if not R[src0] != R[src1]
20: JIN: indirect jump to address R[src0]

Each jump instruction saves the address form which it jumped to register r7 (only if the jump was taken).

24: HLT: halt execution

Execution starts at PC 0. PC increments by 1 after every instruction except in the case a jump is taken, in which case PC is set to the jump target address. Execution after address 0xffff resumes at address 0.

Building a High Level ISA simulator

Write a high level ISA simulator for the SP architecture names iss. It should accept a text file containing a memory dump of the SRAM, each line containing 8 hexadecimal digits, simulate the program till HALT is encountered, and generate a trace file as follows:

```
--- instruction 0 (0000) @ PC 0 (0000) -----------------------------------------------------------
pc = 0000, inst = 0088000f, opcode = 0 (ADD), dst = 2, src0 = 1, src1 = 0, immediate = 0000000f
r[0] = 00000000 r[1] = 0000000f r[2] = 00000000 r[3] = 00000000
r[4] = 00000000 r[5] = 00000000 r[6] = 00000000 r[7] = 00000000

>>>> EXEC: R[2] = 15 ADD 0 <<<<

--- instruction 1 (0001) @ PC 1 (0001) -----------------------------------------------------------
pc = 0001, inst = 01480014, opcode = 0 (ADD), dst = 5, src0 = 1, src1 = 0, immediate = 00000014
r[0] = 00000000 r[1] = 00000014 r[2] = 0000000f r[3] = 00000000
r[4] = 00000000 r[5] = 00000000 r[6] = 00000000 r[7] = 00000000

>>>> EXEC: R[5] = 20 ADD 0 <<<<
```

An example trace is available in lab1_example.zip, as well as a simple assembler. The above trace is for the first two commands of the following program:

```
        asm_cmd(ADD, 2, 1, 0, 15); // 0: R2 = 15
        asm_cmd(ADD, 5, 1, 0, 20); // 1: R5 = 20
        asm_cmd(ADD, 3, 1, 0, 0); // 2:
        asm_cmd(LD,  4, 0, 2, 0); // 3:
        asm_cmd(ADD, 3, 3, 4, 0); // 4:
        asm_cmd(ADD, 2, 2, 1, 1); // 5:
        asm_cmd(JLT, 0, 2, 5, 3); // 6:
        asm_cmd(ST,  0, 3, 1, 15); // 7:
        asm_cmd(HLT, 0, 0, 0, 0); // 8:
       for (i = 0; i < 5; i++)
               mem[15+i] = i;
```

The above program is embedded in the asm.c assembler. Run the assembler as “asm example.bin”, which will generate the following memory dump, that will be an input to the simulator, which will generate the trace:

```
0088000f
01480014
00c80000
11020000
00dc0000
00910001
20150003
1219000f
30000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000001
00000002
00000003
00000004
```

1. Answer the following questions:

1. Write the command which will load the constant 5 into register 2.
2. Write the command to load the constant (-5) into register 2.
3. Write the command to calculate R2 – R3, and save it into R2.
4. It seems as if several jump conditions are missing, for example > and >=. Explain how to implement them using the available opcodes.
5. Explain how to load a 32 bit constant into a register.
6. Explain how subroutine calls can be implemented on this processor.

2. Example program

Refer to the example program in asm.c.

1. What does it compute?
2. Where are the inputs stored?
3. Where are the outputs stored?
4. Submit a commented version of the assembly program, which contains comments in pseudo code explaining what each line does (the first two lines were commented for you).

3. ISS simulator Testing #1

Write a program which will multiply two (possibly signed) numbers. The two input numbers are stored at addresses 1000 and 1001, and the result will be written to 1002. Test with the ISS, and submit the program, trace file, and memory contents 1000-1002.

4. ISS simulator Testing #2

Write a program which will calculate the multiplication table, and write it to memory starting at address 2000. Test it with the ISS simulator, and submit: 1. the program, 2. the trace file, and 3. the memory contents from addresses 2000 – 2099.

5. ISS simulator Testing #3

Submit your code for the ISS simulator, it should compile using "make", producing an executable iss, which can be run as:

```
iss code.bin
```

Make sure to handle corner cases so that your simulator doesn't crash or exit under any circumstance except reaching the HLT instruction. You're free to implement undefined behavior in a way which will be easiest from the hardware point of view.

Trace file should be stored to the file trace.txt. Your ISS simulator will be tested with test programs (you don't get their source code) to verify its correctness.