Essential Skills (6) Computer Architecture

Tykushin Anatoly, SNE

October 2016

Contents

1 Digital circuit

- **Draw a digital circuit of an 4-bit incrementer, i.e., a circuit that satisfies the equation b = a + 1 mod 16. You can use the operations from the set {and,or,nand, nor, not, xor}**
- **Try to find an incrementer with the minimal number of operations needed**
- **Try to minimize the depth of the circuit, i.e. the maximal length of any path from input to output [Optional]**

To sum up all tasks I've developed minimal incrementer, using 6 elements. The circuit was synthesized in QuartusII 9.1 Web Edition CAD. It is shown on the figure 1.

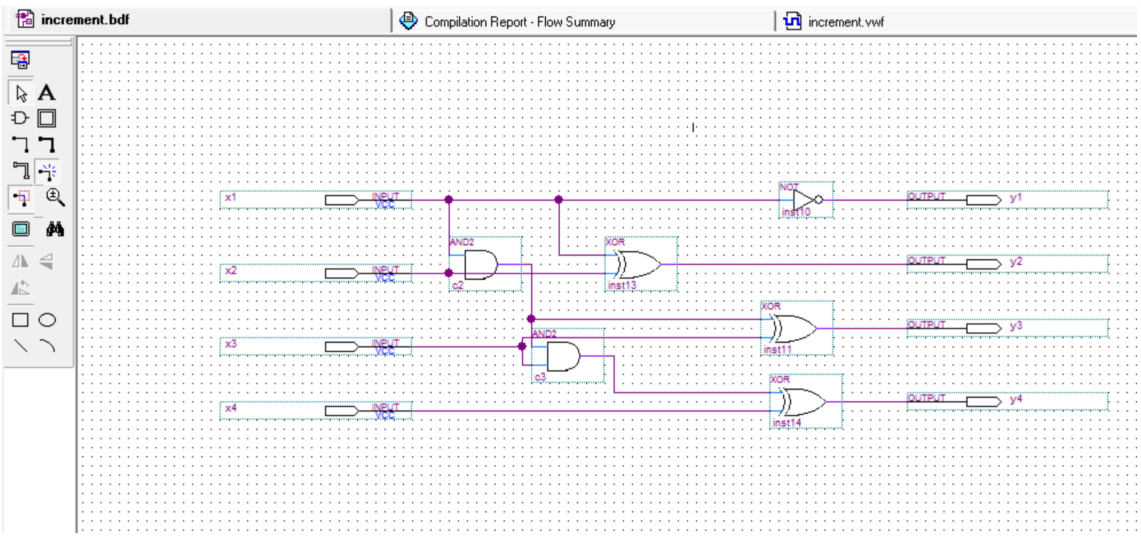


Figure 1: Synthesized incrementer circuit

To prove that drawn circuit works I've simulated it on FPGA (Altera Cyclone). The result is shown on the figure 2. On the top of the picture you can see group called **inp**. It contains $x_4 ... x_1$ pins (input pins). On the output group you can see the values of the $y_4 ... y_1$ pins. Also you can see phenomena called "edge race" - it is a kind of pulses between stable states (in the output e.g. between 1 and 2 and so on...). This was happened because lack of syncronization implementation in this circuit.

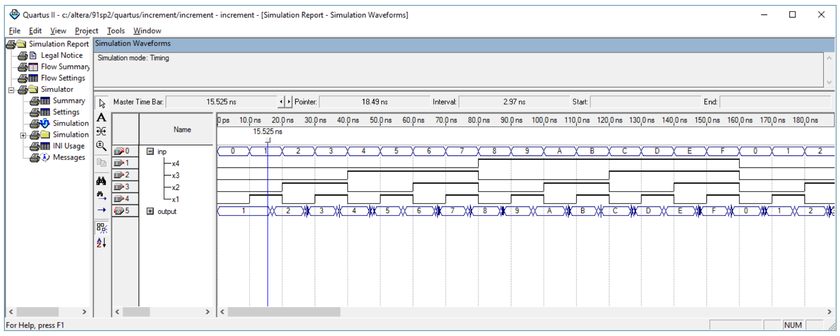


Figure 2: Simulated circuit

2 Experiment Performance measurement

2.1 Parameter measurement

The result of completing this section is shown in the table 2.1

Architecture	Program name	# of instructions	CPI	CPU time
Single cycle	addition.wasm	6	1	6
Single cycle	forloop.wasm	48	1	48
Single cycle	squares.wasm	100	1	100
Multi cycle	addition.wasm	6	3,5	21
Multi cycle	forloop.wasm	49	3,5	172
Multi cycle	squares.wasm	100	3,5	350
Pipeline	addition.wasm	15	1	15
Pipeline	forloop.wasm	98	1	98
Pipeline	squares.wasm	298	1	298

2.2 Explaining things

Which architecture performed the best for every executable and explain why.
According to figure 3 you can see that **pipeline** has the best performance because it completes command in nearly parallel (by delaying loading next command in 1 clock cycle). Here you can see the review of every processor design technology.

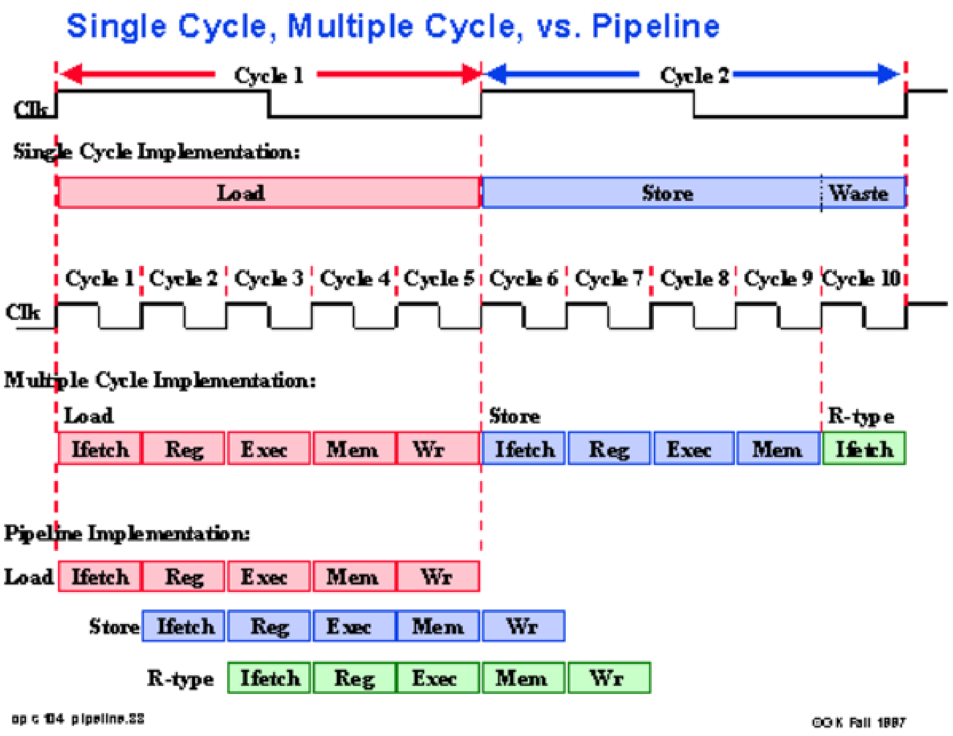


Figure 3: Comparing performance (link)

- Single cycle
  - Advantages
    - Single Cycle per instruction make logic and clock simple
  - Disadvantages
    - Since instructions take different time to finish, memory and functional unit are not efficiently utilized
    - Cycle time is the longest delay (load instruction)
    - Best possible CPI is 1 (however, lower MIPS and longer clock period (lower clock frequency); hence, lower performance)
- Multi cycle
  - Advantages
    - Better MIPS and smaller clock period (higher clock frequency)
    - Hence, better performance than Single Cycle processor
  - Disadvantages
    - Higher CPI than single cycle processor
- Pipeline
  - principles
    - Break instructions across multiple clock cycles
    - Design a separare stage for the execution performed during each clock cycle
    - Add pipeline registers (flip-flops) to isolate signals between different stages

A program counter is present in the architecture. Can you explain what the program counter function is?

Program counter (PC) is a command counter, it points to the start of the next instruction which will run next after this. It provides sequential execution of commands.

What do you notice about the behavior when running a simulation?

Explain instructions: behaviour and usage

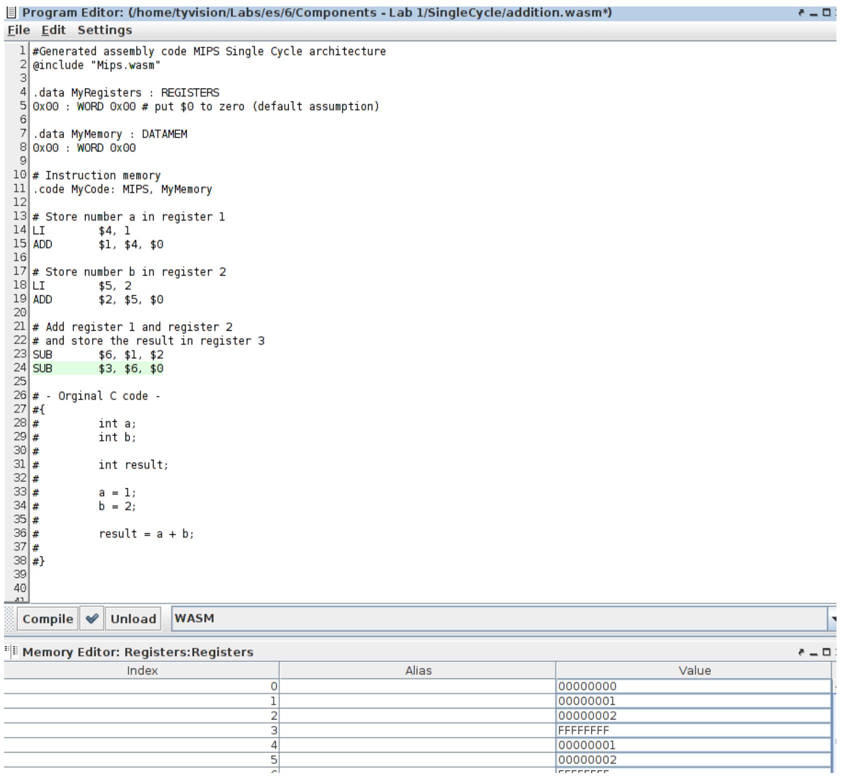
- ADDI
- NOP
- LI
- BNE

For better perspicuity I've collected information about instructions in a table 2.2.

Command	Preference	Description
ADDI	Description	Adds a register and a sign-extended immediate value and stores the result in a register
	Operation	\$t = \$s + imm; advancepc (4)
	Syntax	addi \$t, \$s, imm
NOOP	Description	Performs no operation
	Operation	advancepc (4)
	Syntax	noop
LI	Description	Loads data to a register
	Operation	Load immediate into to register s, i.e. R[s] = immed. The way this is translated depends on whether immed is 16 bits or 32 bits
	Syntax	LI \$rs, addr
BNE	Description	Branches if the two registers are not equal
	Operation	if \$s != \$t advancepc (offset << 2)); else advancepc (4);
	Syntax	bne \$s, \$t, offset

Now change the first program to subtract two numbers, do not change the initialization of the variables can you explain what happens

All ADD commands was replaced by SUB commands. The result is shown on the figure 4. The result is 0xFFFFFFFF (as it is an unsigned operation - overflow).


Program Editor: (/home/tyvision/Labs/es/6/Components - Lab 1/SingleCycle/addition.wasm*)

```
#Generated assembly code MIPS Single Cycle architecture
@include "Mips.wasm"

.data MyRegisters : REGISTERS
0x00 : WORD 0x00 # put $0 to zero (default assumption)

.data MyMemory : DATAMEM
0x00 : WORD 0x00

# Instruction memory
.code MyCode: MIPS, MyMemory

# Store number a in register 1
LI        $4, 1
ADD       $1, $4, $0

# Store number b in register 2
LI        $5, 2
ADD       $2, $5, $0

# Add register 1 and register 2
# and store the result in register 3
SUB       $6, $1, $2
SUB       $3, $6, $0

# - Orginal C code -
#{
#         int a;
#         int b;
#
#         int result;
#
#         a = 1;
#         b = 2;
#
#         result = a + b;
#
#}
```

Memory Editor: Registers:Registers

Index	Alias	Value
0		00000000
1		00000001
2		00000002
3		FFFFFFFF
4		00000001
5		00000002

Figure 4: Subtraction

Finally explain why the clock rate (frequency) of a cpu is not a good benchmark for the performance of a cpu.

Firstly, CPU performance is a multiplication of 3 parameters: instructions executed, CPI, clock cycle time.

So, frequency is just the length of a cycle. Generally, higher frequency is better. But as we can see on example of single cycle technology, this is unefficient. For example, length of loading and executing latency must be different. Much more important is how processor uses this cycles for making equations, performing operations.