마이크로프로세서
-Execution Time -

Daejin Park

School of Electronics Engineering, KNU, KOREA

2019.05.08, 05.10

ACES Lab.

Instructions

숙제

	4	4	8		
MOV0	0000	R_n	direct	$R_n \leftarrow Mem\{direct\}$ $PC \leftarrow PC + 1$	8clks
MOV1	0001	R_n	direct	$Mem\{direct\} \leftarrow R_n$ $PC \leftarrow PC + 1$	8clks
MOV2	0010	R_n	0000 R_m	$Mem\{\ R_n\ \} \leftarrow R_m$ $PC \leftarrow PC + 1$	12clks
MOV3	0011	R_n	#immed	$R_n \leftarrow \#immed$ $PC \leftarrow PC + 1$	6clks

Instructions

ADD

4	4	8	
0100	R_n	R_m	0000

$R_n \leftarrow R_n + (R_m \gg 4)$
$PC \leftarrow PC + 1$ 4clks

SUB

4	4	8	
0101	R_n	R_m	0000

$R_n \leftarrow R_n - (R_m \gg 4)$
$PC \leftarrow PC + 1$ 4clks

JZ

4	4	8
0110	R_n	(signed) relative

If R_n == 0 12clks
$PC \leftarrow PC + 1 + relative$

MUL

4	4	8	
0111	R_n	R_m	0000

$R_n \leftarrow R_n * R_m$ 30clks
$PC \leftarrow PC + 1$

숙제

MOV4

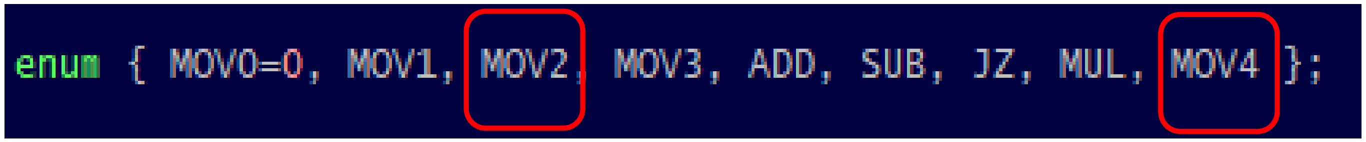
4	4	8	
1000	R_n	R_m	0000

$R_n \leftarrow Mem\{R_m\}$ 2clks
$PC \leftarrow PC + 1$

ACES Lab.

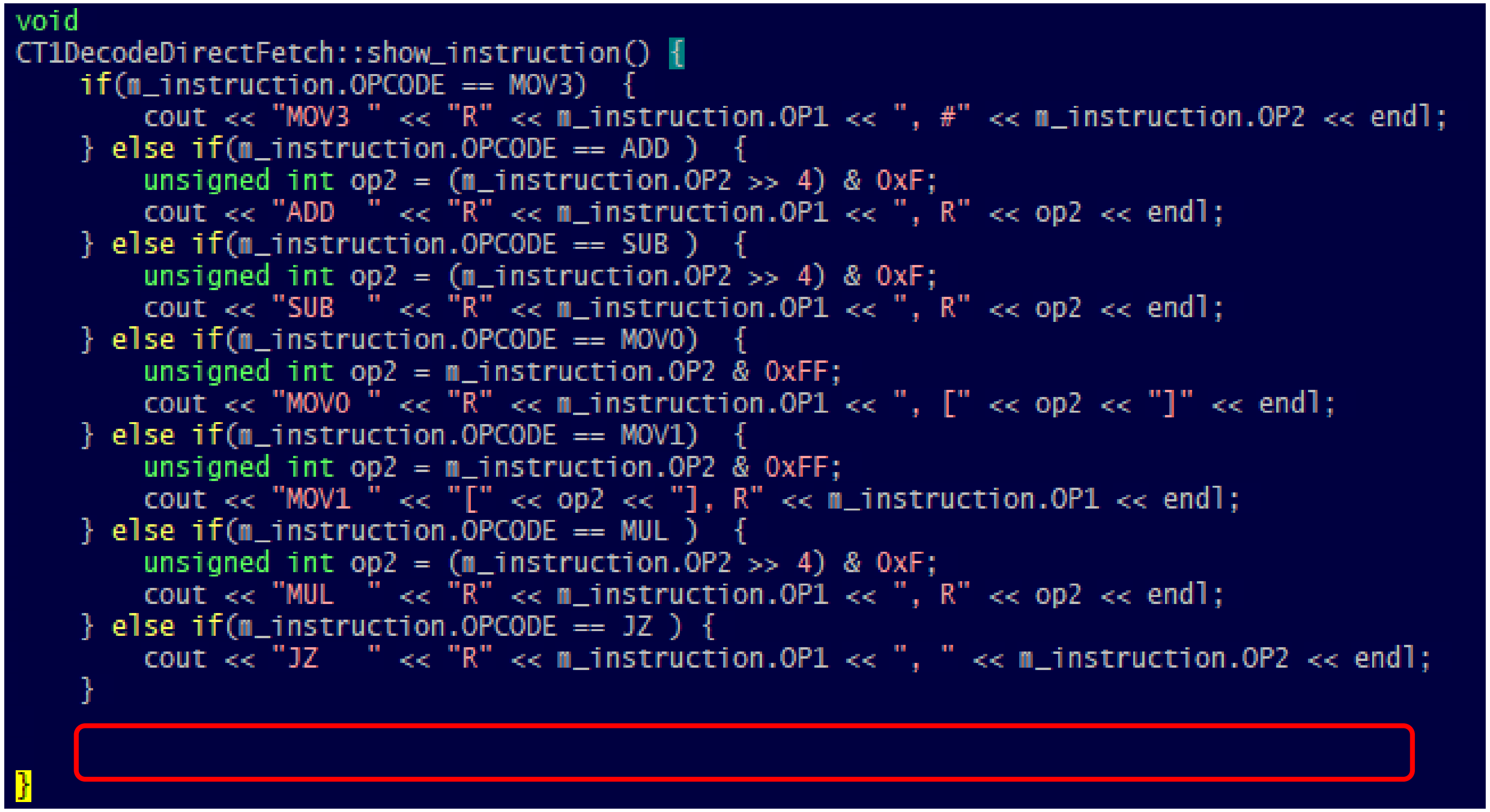
Decode 일부수정

CDecode.h



```
enum { MOV0=0, MOV1, MOV2, MOV3, ADD, SUB, JZ, MUL, MOV4 };
```

CDecode.cpp



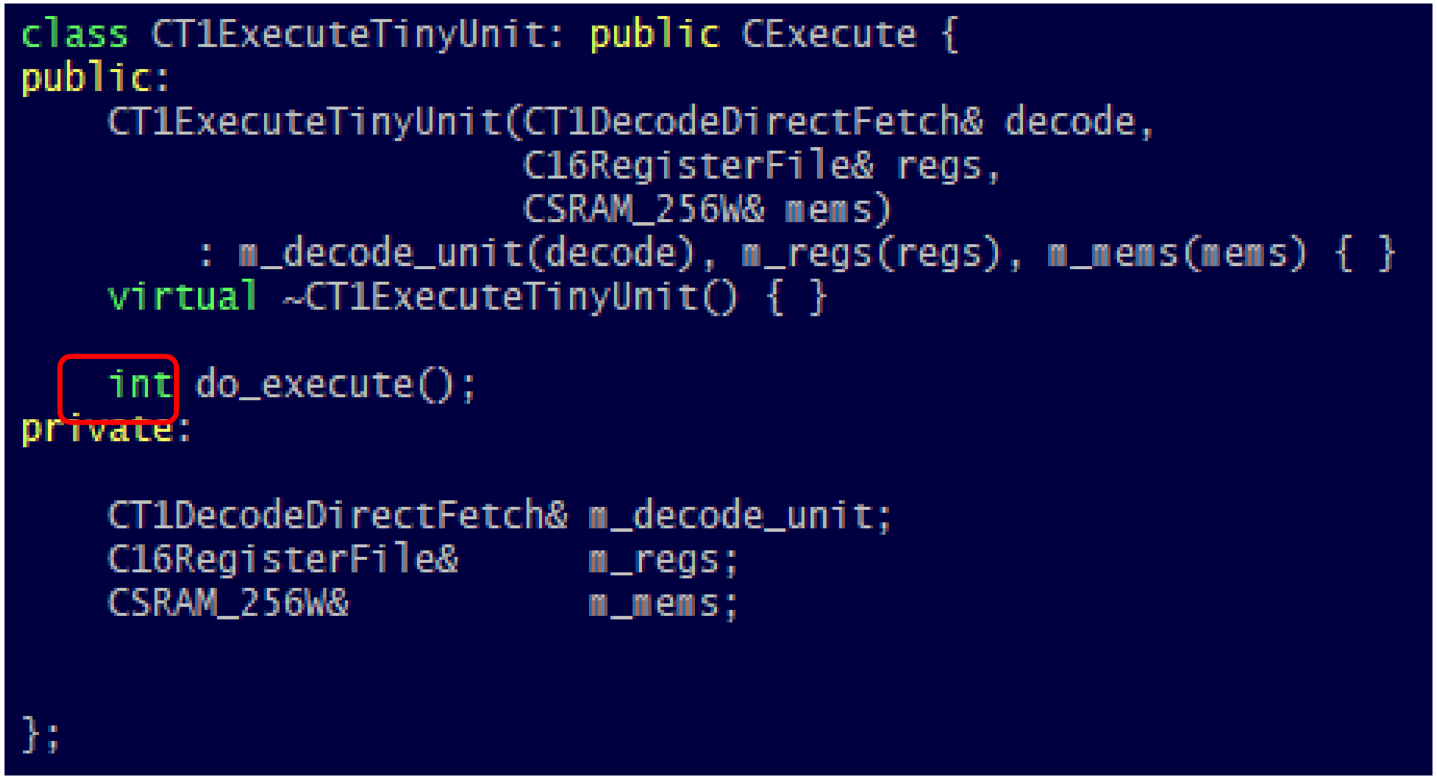
```
void
CT1DecodeDirectFetch::show_instruction() {
    if(m_instruction.OPCODE == MOV3)  {
        cout << "MOV3 " << "R" << m_instruction.OP1 << ", #" << m_instruction.OP2 << endl;
    } else if(m_instruction.OPCODE == ADD )  {
        unsigned int op2 = (m_instruction.OP2 >> 4) & 0xF;
        cout << "ADD  " << "R" << m_instruction.OP1 << ", R" << op2 << endl;
    } else if(m_instruction.OPCODE == SUB )  {
        unsigned int op2 = (m_instruction.OP2 >> 4) & 0xF;
        cout << "SUB  " << "R" << m_instruction.OP1 << ", R" << op2 << endl;
    } else if(m_instruction.OPCODE == MOV0)  {
        unsigned int op2 = m_instruction.OP2 & 0xFF;
        cout << "MOV0 " << "R" << m_instruction.OP1 << ", [" << op2 << "]" << endl;
    } else if(m_instruction.OPCODE == MOV1)  {
        unsigned int op2 = m_instruction.OP2 & 0xFF;
        cout << "MOV1 " << "[" << op2 << "], R" << m_instruction.OP1 << endl;
    } else if(m_instruction.OPCODE == MUL )  {
        unsigned int op2 = (m_instruction.OP2 >> 4) & 0xF;
        cout << "MUL  " << "R" << m_instruction.OP1 << ", R" << op2 << endl;
    } else if(m_instruction.OPCODE == JZ ) {
        cout << "JZ   " << "R" << m_instruction.OP1 << ", " << m_instruction.OP2 << endl;
    }


}
```

MOV2, MOV4 추가 (숙제)

ACES Lab.

Execution시 수행시간 측정

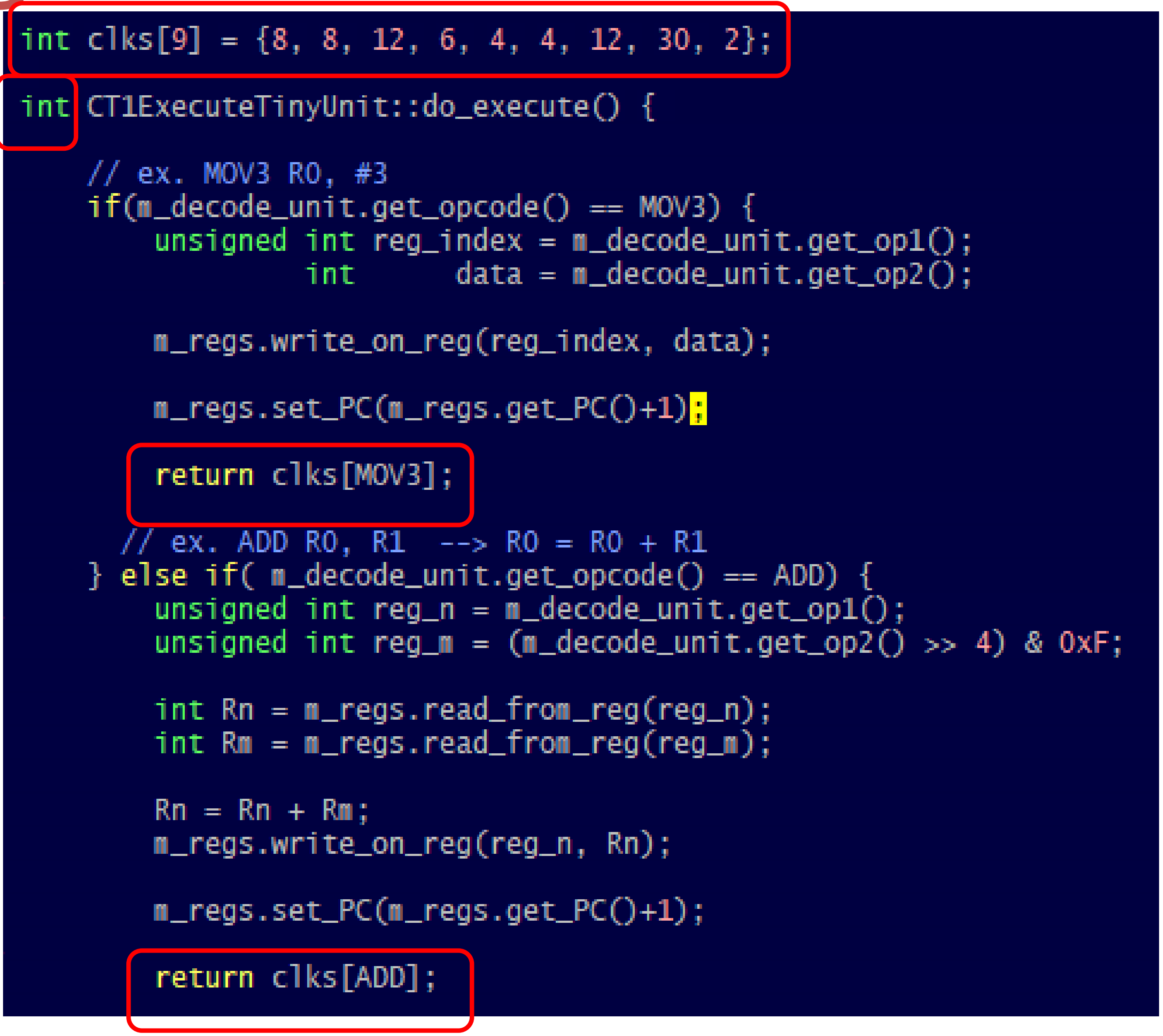
수정



```
class CT1ExecuteTinyUnit: public CExecute {
public:
    CT1ExecuteTinyUnit(CT1DecodeDirectFetch& decode,
                       C16RegisterFile& regs,
                       CSRAM_256W& mems)
        : m_decode_unit(decode), m_regs(regs), m_mems(mems) { }
    virtual ~CT1ExecuteTinyUnit() { }

    int do_execute();
private:

    CT1DecodeDirectFetch& m_decode_unit;
    C16RegisterFile&      m_regs;
    CSRAM_256W&           m_mems;


};
```

Execute 수정

CExecute.cpp

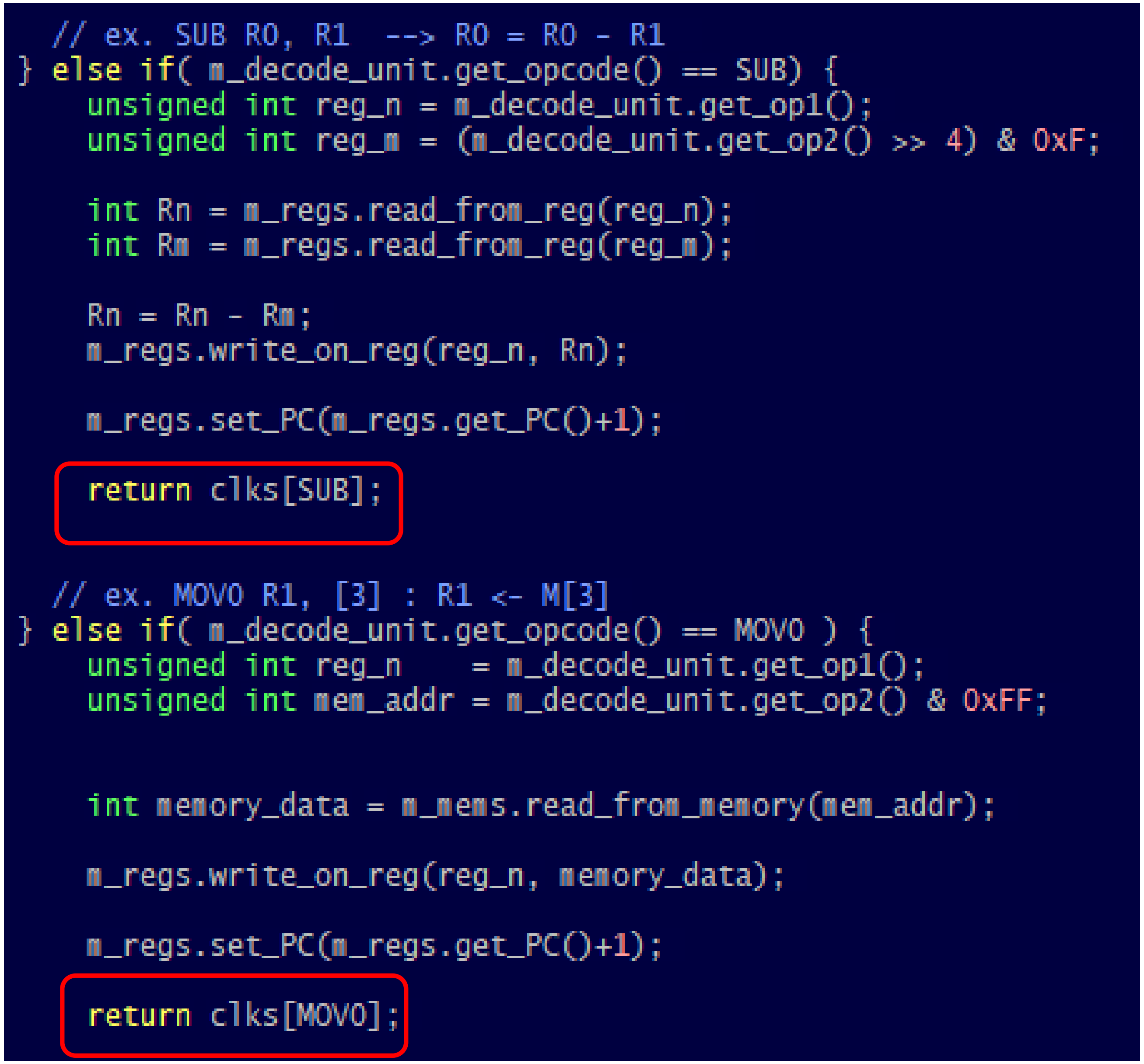


```
int clks[9] = {8, 8, 12, 6, 4, 4, 12, 30, 2};
int CT1ExecuteTinyUnit::do_execute() {

    // ex. MOV3 R0, #3
    if(m_decode_unit.get_opcode() == MOV3) {
        unsigned int reg_index = m_decode_unit.get_op1();
                 int      data = m_decode_unit.get_op2();

        m_regs.write_on_reg(reg_index, data);

        m_regs.set_PC(m_regs.get_PC()+1);

        return clks[MOV3];

      // ex. ADD R0, R1   --> R0 = R0 + R1
    } else if( m_decode_unit.get_opcode() == ADD) {
        unsigned int reg_n = m_decode_unit.get_op1();
        unsigned int reg_m = (m_decode_unit.get_op2() >> 4) & 0xF;

        int Rn = m_regs.read_from_reg(reg_n);
        int Rm = m_regs.read_from_reg(reg_m);

        Rn = Rn + Rm;
        m_regs.write_on_reg(reg_n, Rn);

        m_regs.set_PC(m_regs.get_PC()+1);

        return clks[ADD];
```

ACES Lab.

Execute 수정

CExecute.cpp

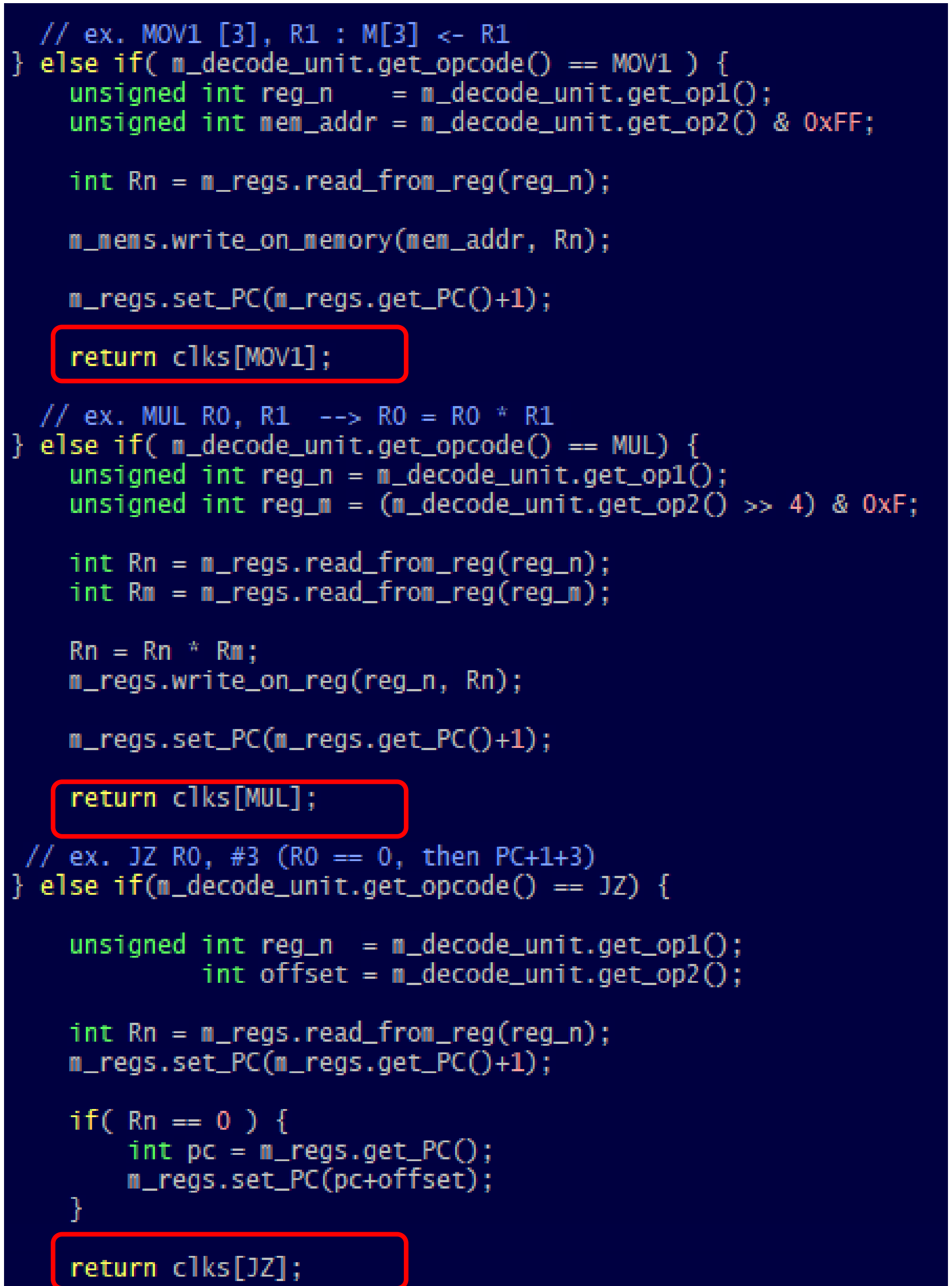


```
   // ex. SUB R0, R1  --> R0 = R0 - R1
} else if( m_decode_unit.get_opcode() == SUB) {
    unsigned int reg_n = m_decode_unit.get_op1();
    unsigned int reg_m = (m_decode_unit.get_op2() >> 4) & 0xF;

    int Rn = m_regs.read_from_reg(reg_n);
    int Rm = m_regs.read_from_reg(reg_m);

    Rn = Rn - Rm;
    m_regs.write_on_reg(reg_n, Rn);

    m_regs.set_PC(m_regs.get_PC()+1);

    return clks[SUB];

   // ex. MOV0 R1, [3] : R1 <- M[3]
} else if( m_decode_unit.get_opcode() == MOV0 ) {
    unsigned int reg_n    = m_decode_unit.get_op1();
    unsigned int mem_addr = m_decode_unit.get_op2() & 0xFF;


    int memory_data = m_mems.read_from_memory(mem_addr);

    m_regs.write_on_reg(reg_n, memory_data);

    m_regs.set_PC(m_regs.get_PC()+1);

    return clks[MOV0];
```

ACES Lab.

Execute 수정

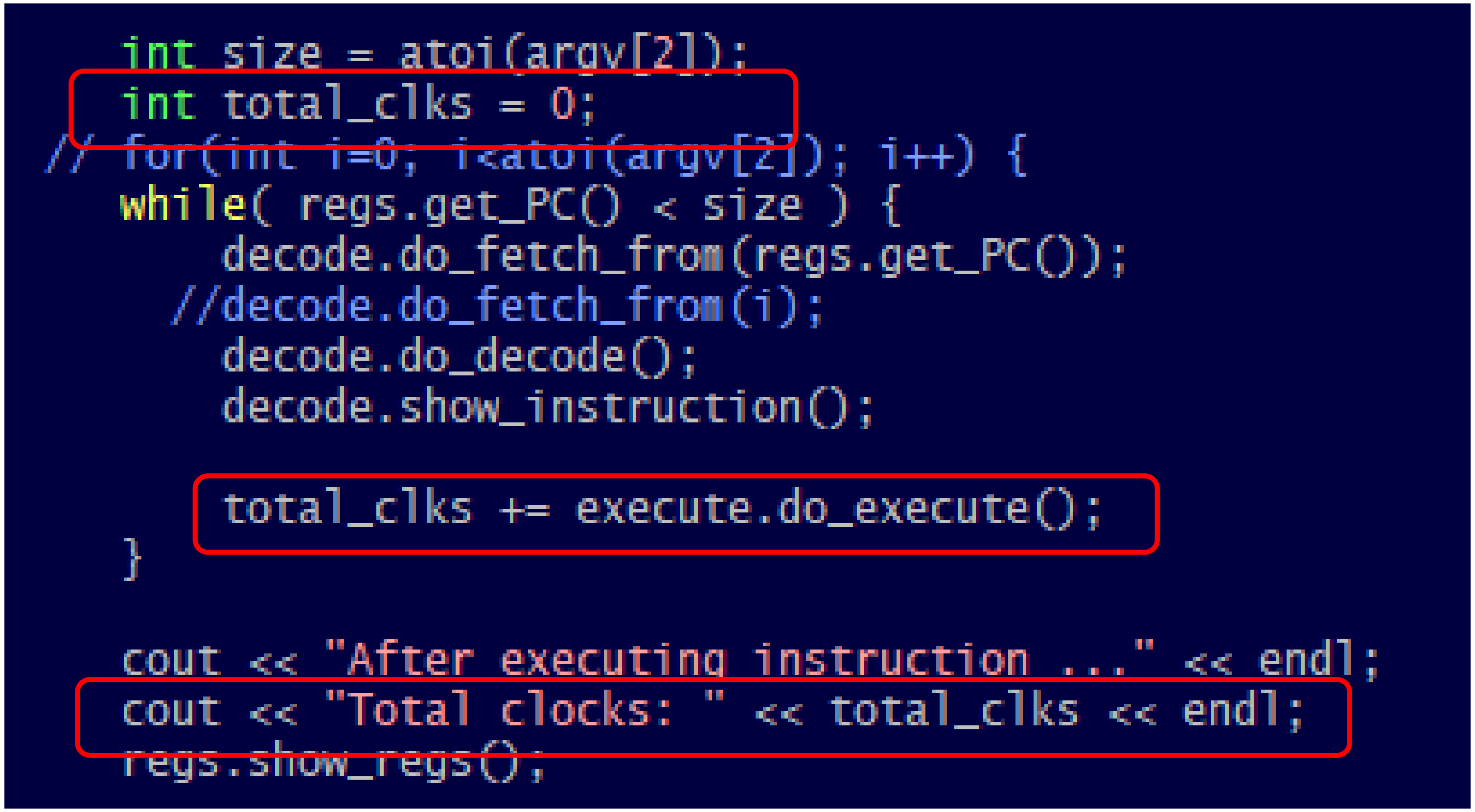
CExecute.cpp



```
  // ex. MOV1 [3], R1 : M[3] <- R1
} else if( m_decode_unit.get_opcode() == MOV1 ) {
    unsigned int reg_n    = m_decode_unit.get_op1();
    unsigned int mem_addr = m_decode_unit.get_op2() & 0xFF;

    int Rn = m_regs.read_from_reg(reg_n);

    m_mems.write_on_memory(mem_addr, Rn);

    m_regs.set_PC(m_regs.get_PC()+1);

    return clks[MOV1];

  // ex. MUL R0, R1  --> R0 = R0 * R1
} else if( m_decode_unit.get_opcode() == MUL) {
    unsigned int reg_n = m_decode_unit.get_op1();
    unsigned int reg_m = (m_decode_unit.get_op2() >> 4) & 0xF;

    int Rn = m_regs.read_from_reg(reg_n);
    int Rm = m_regs.read_from_reg(reg_m);

    Rn = Rn * Rm;
    m_regs.write_on_reg(reg_n, Rn);

    m_regs.set_PC(m_regs.get_PC()+1);

    return clks[MUL];

 // ex. JZ R0, #3 (R0 == 0, then PC+1+3)
} else if(m_decode_unit.get_opcode() == JZ) {

    unsigned int reg_n  = m_decode_unit.get_op1();
             int offset = m_decode_unit.get_op2();

    int Rn = m_regs.read_from_reg(reg_n);
    m_regs.set_PC(m_regs.get_PC()+1);

    if( Rn == 0 ) {
        int pc = m_regs.get_PC();
        m_regs.set_PC(pc+offset);
    }

    return clks[JZ];
```

main 수정

main.cpp

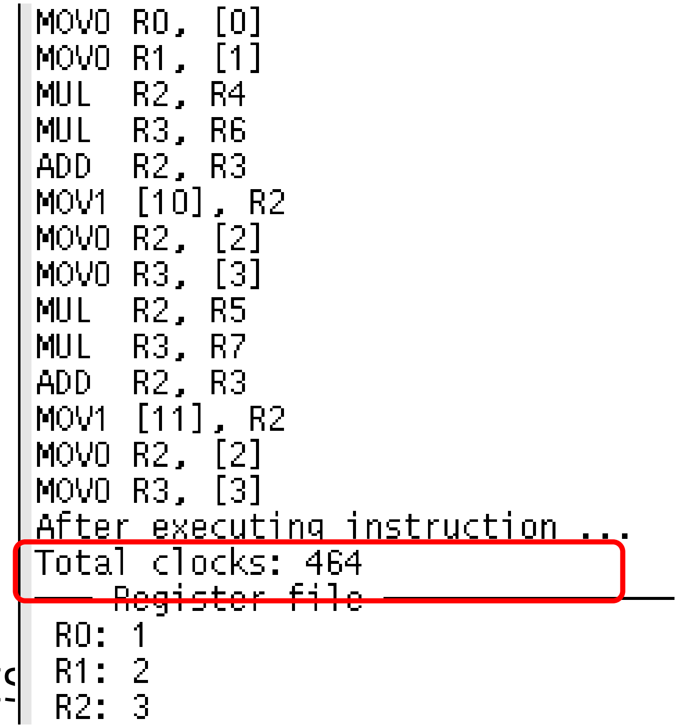


```
    int size = atoi(argv[2]);
    int total_clks = 0;
//  for(int i=0; i<atoi(argv[2]); i++) {
    while( regs.get_PC() < size ) {
        decode.do_fetch_from(regs.get_PC());
      //decode.do_fetch_from(i);
        decode.do_decode();
        decode.show_instruction();

        total_clks += execute.do_execute();
    }

    cout << "After executing instruction ..." << endl;
    cout << "Total clocks: " << total_clks << endl;
    regs.show_regs();
```

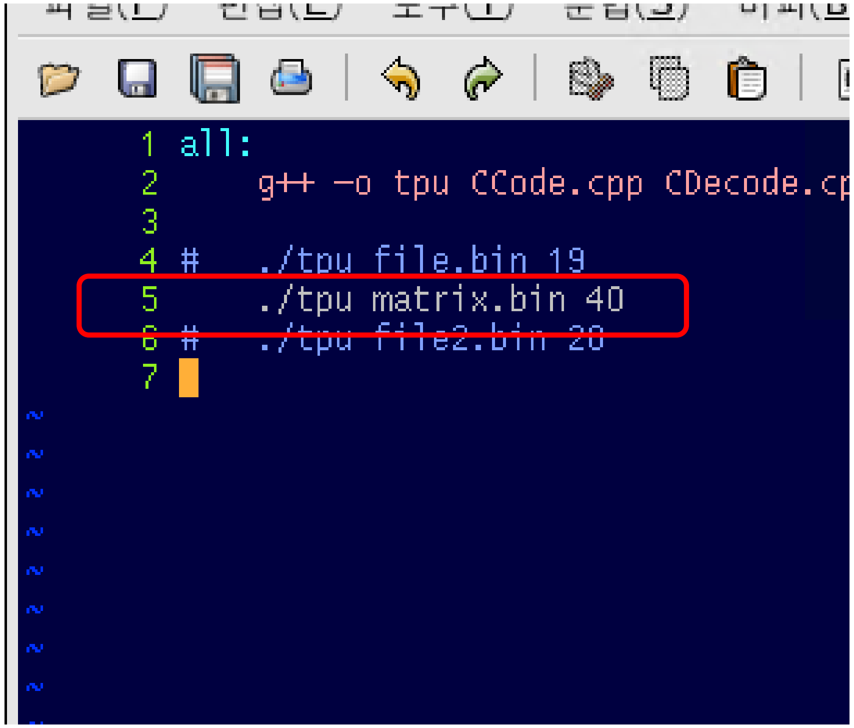
ACES Lab.

명령어 라인수 vs. 실행시간

- 명령어 라인수는 정적인 특징
  - 라인 길이는 실제 프로그램의 수행시간이 아님.
- 실행시간 = 클럭수 * 1/freq
  - 클럭수는 마이크로프로세서가 코드를 읽어들여 해석하여 실행함으로써 프로그램 흐름에 의해 결정되는 동적인 특성임.

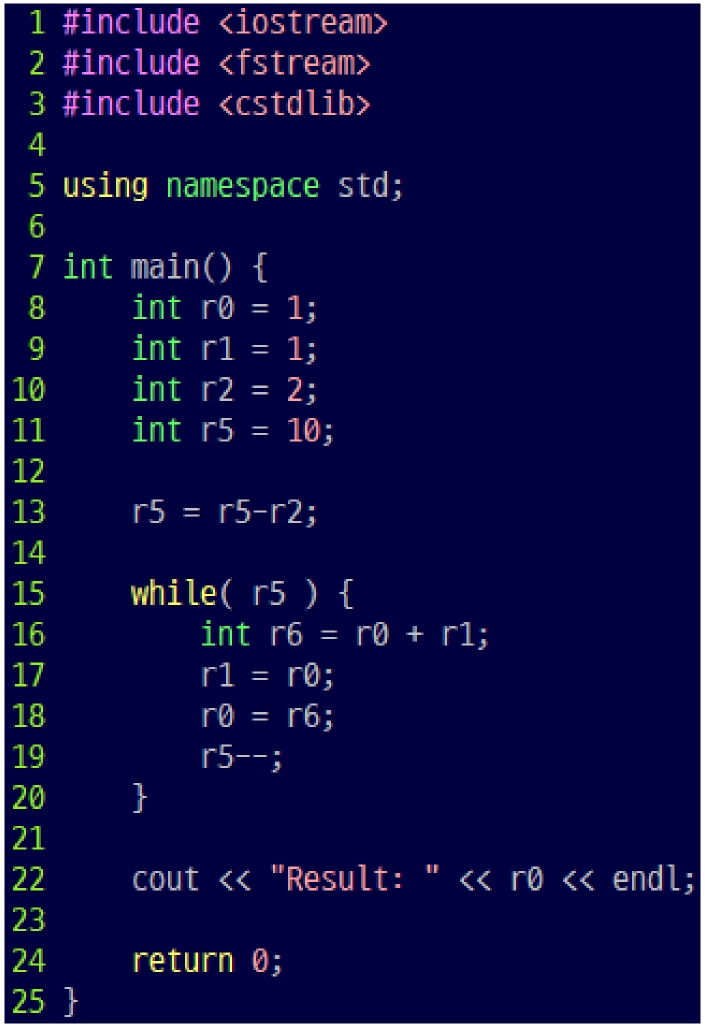


```
MOV0  R0,  [0]
MOV0  R1,  [1]
MUL   R2,  R4
MUL   R3,  R6
ADD   R2,  R3
MOV1  [10], R2
MOV0  R2,  [2]
MOV0  R3,  [3]
MUL   R2,  R5
MUL   R3,  R7
ADD   R2,  R3
MOV1  [11], R2
MOV0  R2,  [2]
MOV0  R3,  [3]
After executing instruction ...
Total clocks: 464
---- Register file -------------
 R0: 1
 R1: 2
 R2: 3
```



```
all:
	g++ -o tpu CCode.cpp CDecode.cp
	
#	./tpu file.bin 19
	./tpu matrix.bin 40
#	./tpu file2.bin 20

~
~
~
~
~
~
~
~
```

숙제2



```
#include <iostream>
#include <fstream>
#include <cstdlib>

using namespace std;

int main() {
    int r0 = 1;
    int r1 = 1;
    int r2 = 2;
    int r5 = 10;

    r5 = r5-r2;

    while( r5 ) {
        int r6 = r0 + r1;
        r1 = r0;
        r0 = r6;
        r5--;
    }

    cout << "Result: " << r0 << endl;

    return 0;
}
```

왼쪽 C코드에 대한 TPU 명령어로 표현하여 성능을 측정한다.

- MOV4를 사용하지 않은 어셈블리 코드를 작성하고 성능 측정
- MOV4를 사용할 때 성능 측정 비교

ACES Lab.

Q & A

Thank you for your attention

Architecture and Compiler
for Embedded Systems Lab.

School of Electronics Engineering, KNU
ACES Lab (boltanut@knu.ac.kr)