마이크로프로세서
-MUL추가, Decoding시 bit masking-

Daejin Park

School of Electronics Engineering, KNU, KOREA

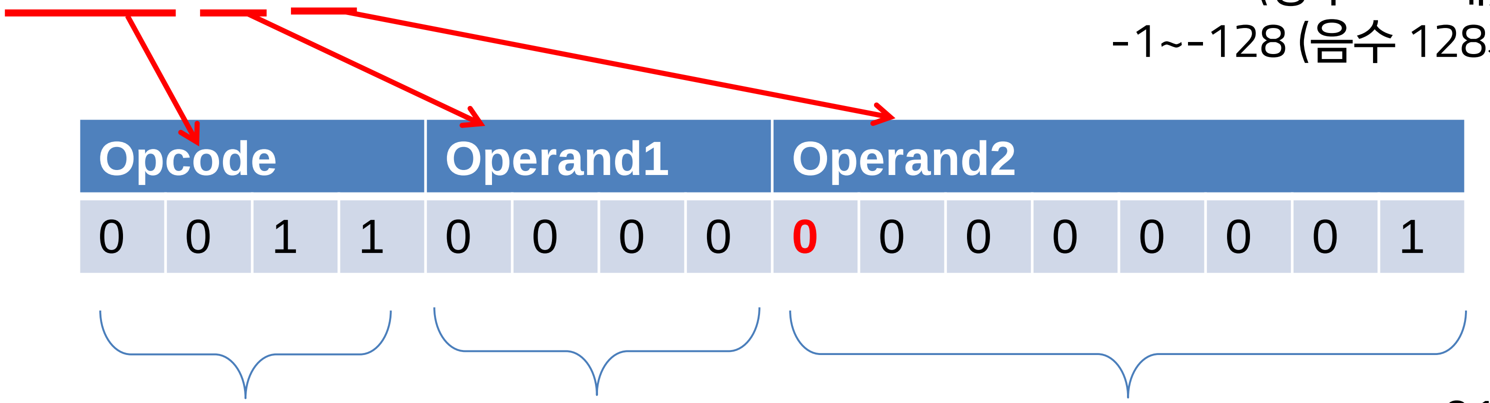
2019.04.12

ACES Lab.

MOV3, #immediate value는 signed값임

- MOV3 R0, #1

Signed값으로 표현한다면
0~127 (양수 128개)
-1~-128 (음수 128개)



Opcode				Operand1				Operand2							
0	0	1	1	0	0	0	0	**0**	0	0	0	0	0	0	1

명령어 종류 16가지 (최대)

16개 레지스터 지정가능

-128~127 값 표현 가능 (데이터)

01111111	127
~	
00000010	2
00000001	1
00000000	0
11111111	-1
11111110	-2
~	
10000001	-127
10000000	-128

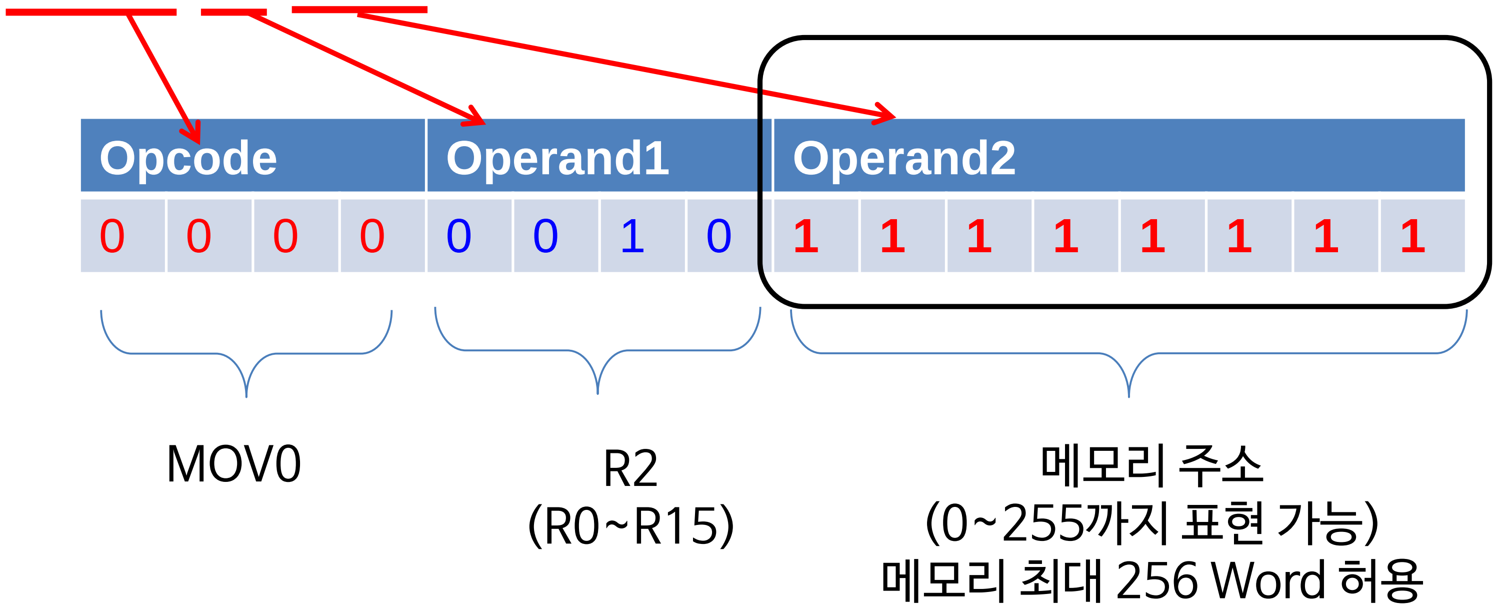
MOV0, MOV1의 Op2는 unsigned값임

- MOV0 Rn, [addr]
  - Rn ← M[addr]

Signed로 표현 시 -1이지만 MOV0, MOV1명령어에서는 unsigne로 표현해야 함 (메모리 주소 0~255값 표현하므로)

→ 하위 8비트만 masking한 뒤 unsigned로 casting 필요

- Ex. MOV0 R2, [**255**]



Decode 일부수정

CDecode.cpp

```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;
    }
}
```

Execute 수정

CExecute.cpp

```cpp
  // 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);

    return true;

  // 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);

    return true;
```

ACES Lab.

ADD, SUB에서 op2는 unsigned 4비트임

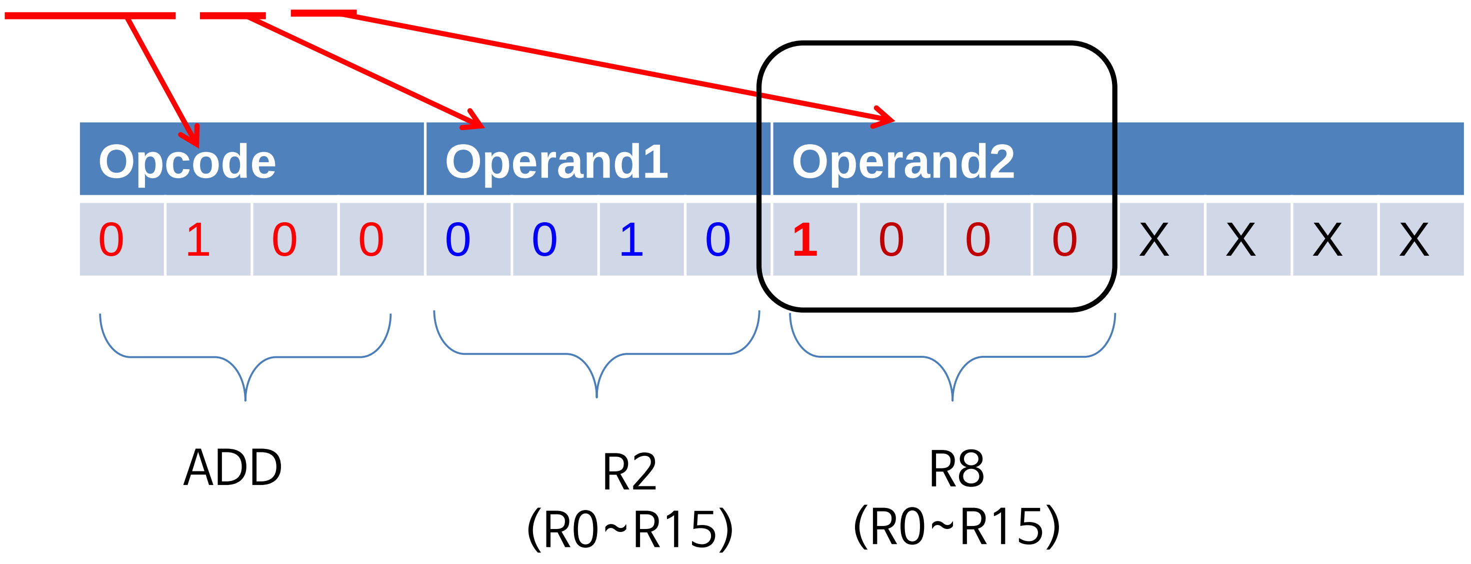
- ADD Rn, Rm
  - Rn = Rn + Rm

1000 >> 4 실행 시

⇨ 11111000이 됨 ⇨ 00001000으로 만들어야 함

ADD, SUB, MUL의 op2는 unsigned임 (0~15)
따라서, 하위 4비트만 masking 해야 함.

- Ex. ADD R2, R8



ACES Lab.

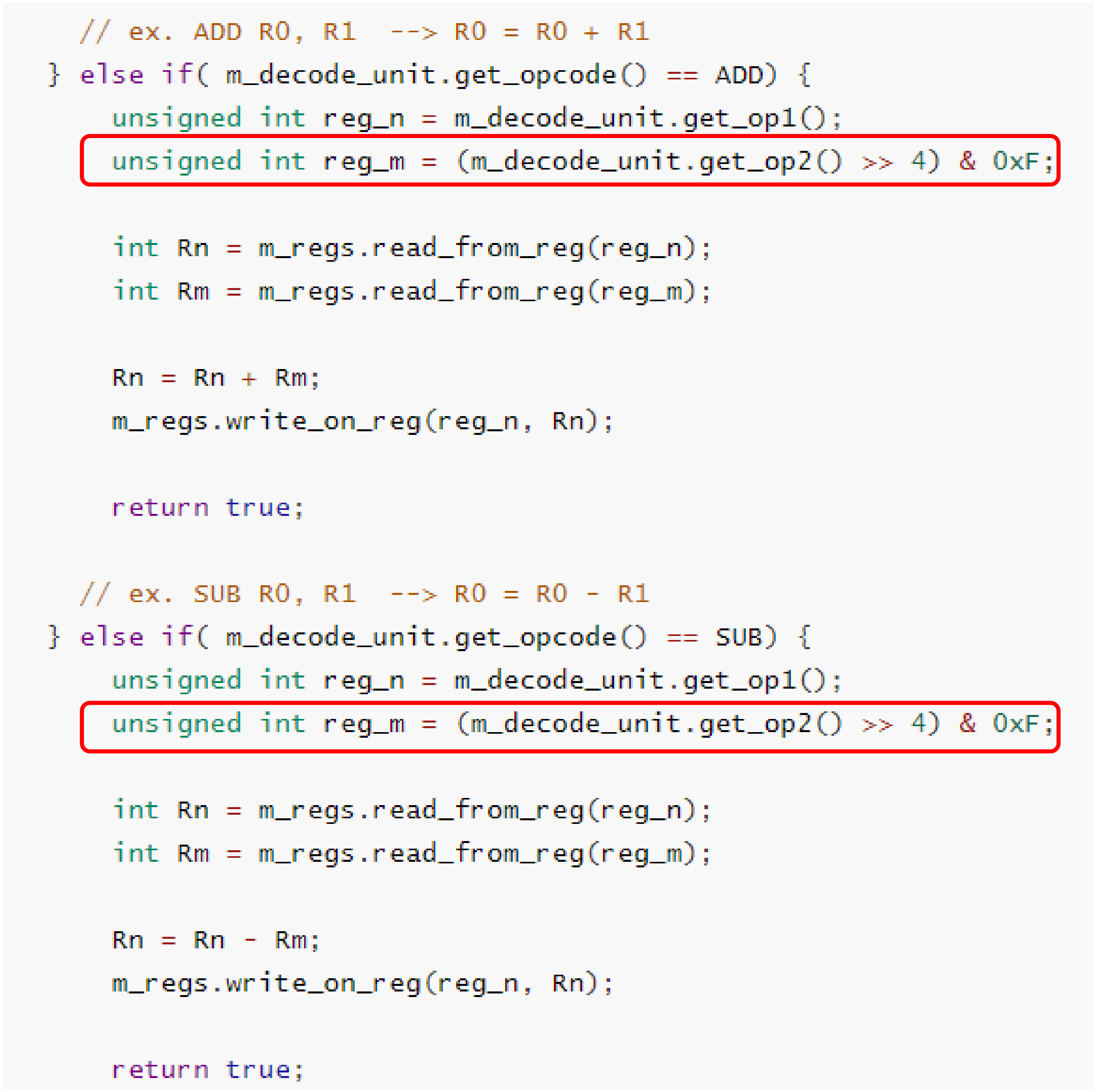
Decode 일부수정

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;
    }
}
```

Execute 수정

CExecute.cpp

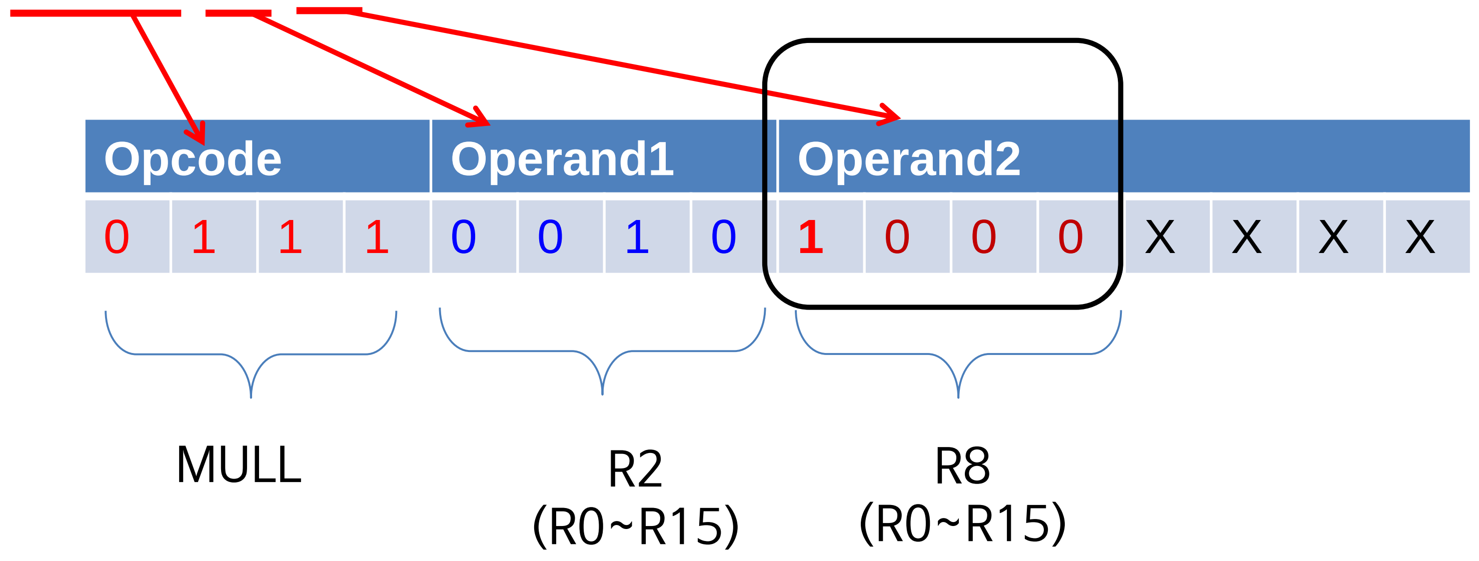


```
    // ex. ADD R0, R1  --> R0 = R0 + R1
  } else if( m_decode_unit.get_opcode() == ADD) {
    unsigned int reg_n = m_decode_unit.get_op1();
    unsigned int reg_m = (m_decode_unit.get_op2() >> 4) & 0xF;

    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);

    return true;

    // 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);

    return true;
```

MUL 추가

- MUL Rn, Rm
  - Rn = Rn * Rm

- Ex. MUL R2, R**8**



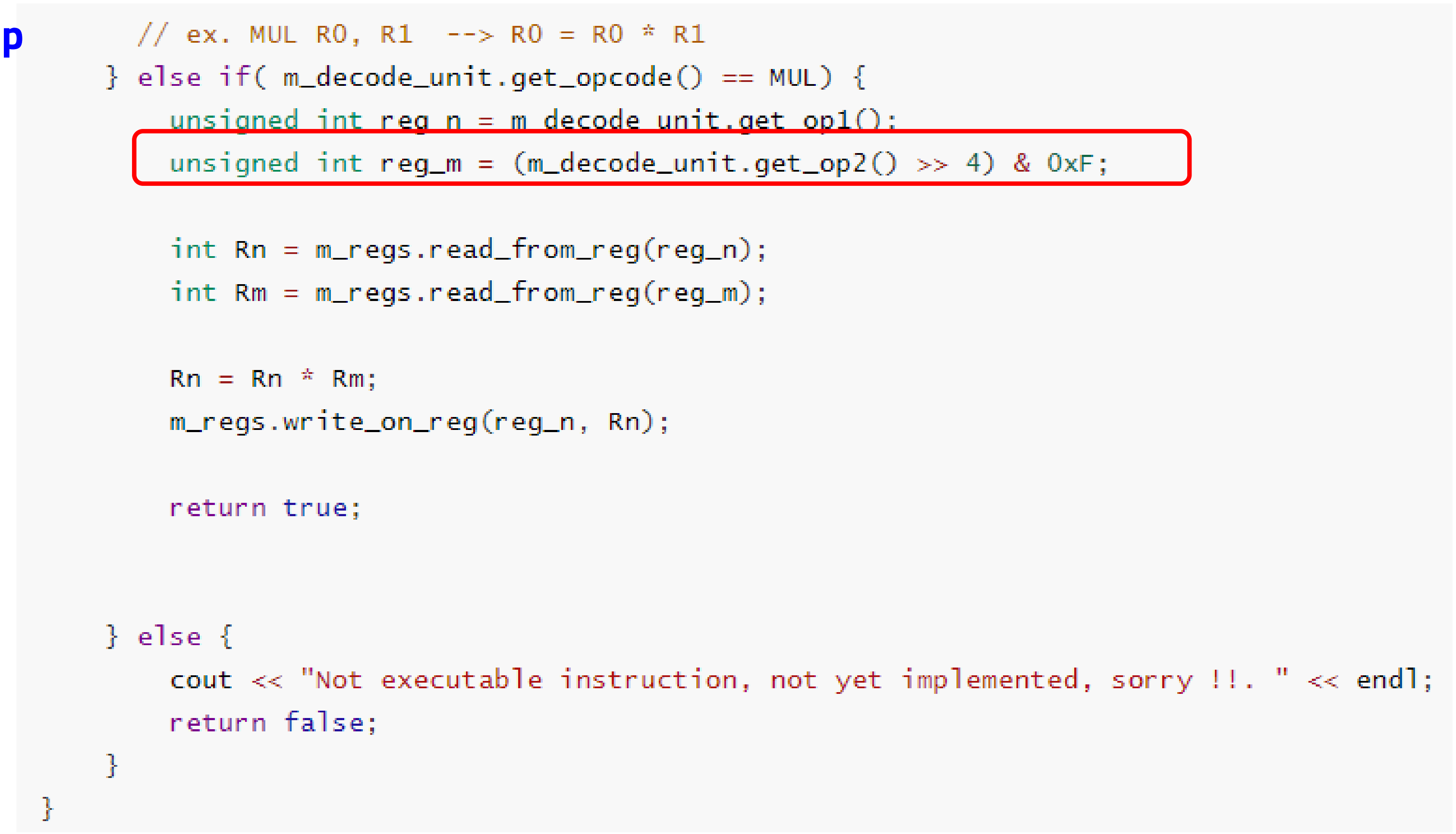
Decode 일부수정

CDecode.cpp

```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;
    }
}
```

Execute 수정

CExecute.cpp



```
        // 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);

        return true;


    } else {
        cout << "Not executable instruction, not yet implemented, sorry !!. " << endl;
        return false;
    }
}
```

CMemory

CMemory.h

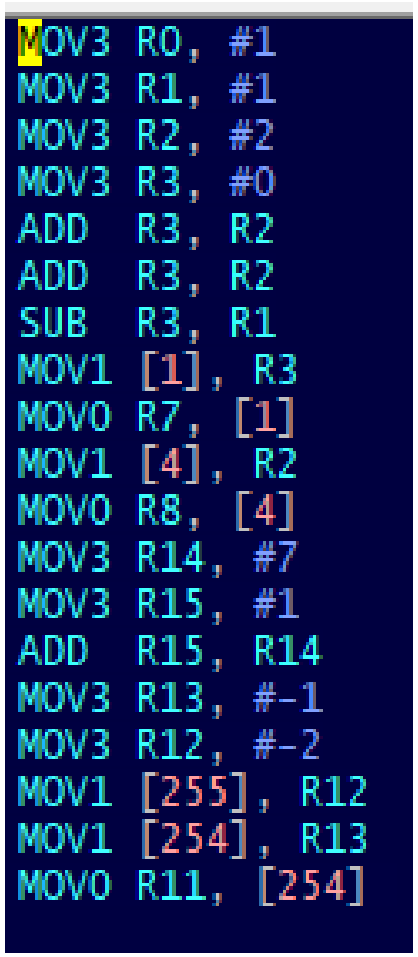
```
class CSRAM_256W : public CMemory {
public:
    CSRAM_256W() { }
    virtual ~CSRAM_256W() { }

    void write_on_memory(unsigned int index, int data) { m_mems[index] = data; }
    int  read_from_memory(unsigned int index)          { return m_mems[index]; }

    void show_mems(unsigned int start_addr, unsigned int end_addr);

private:
    int m_mems[256];

};
```

CMemory.cpp

```
#include "CMemory.h"

void CSRAM_256W::show_mems(unsigned int start_addr, unsigned int end_addr) {
    cout << "--- Memory Dump (addr: " << (int)start_addr << "~" << (int)end_addr << ")"
<< endl;
    for(unsigned int i=start_addr; i<=end_addr; i++) {
        cout << m_mems[i] << " ";
    }
    cout << endl;

}
```

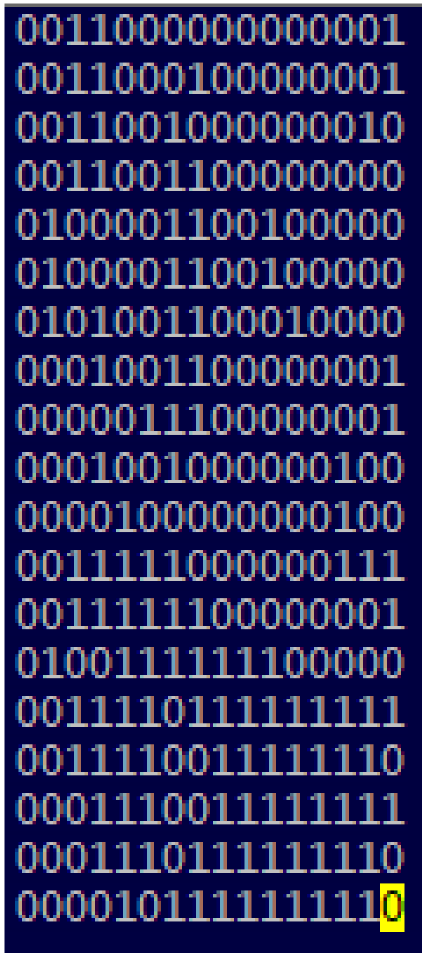
ACES Lab.

테스트 코드



```
MOV3 R0, #1
MOV3 R1, #1
MOV3 R2, #2
MOV3 R3, #0
ADD  R3, R2
ADD  R3, R2
SUB  R3, R1
MOV1 [1], R3
MOV0 R7, [1]
MOV1 [4], R2
MOV0 R8, [4]
MOV3 R14, #7
MOV3 R15, #1
ADD  R15, R14
MOV3 R13, #-1
MOV3 R12, #-2
MOV1 [255], R12
MOV1 [254], R13
MOV0 R11, [254]
```

file.bin

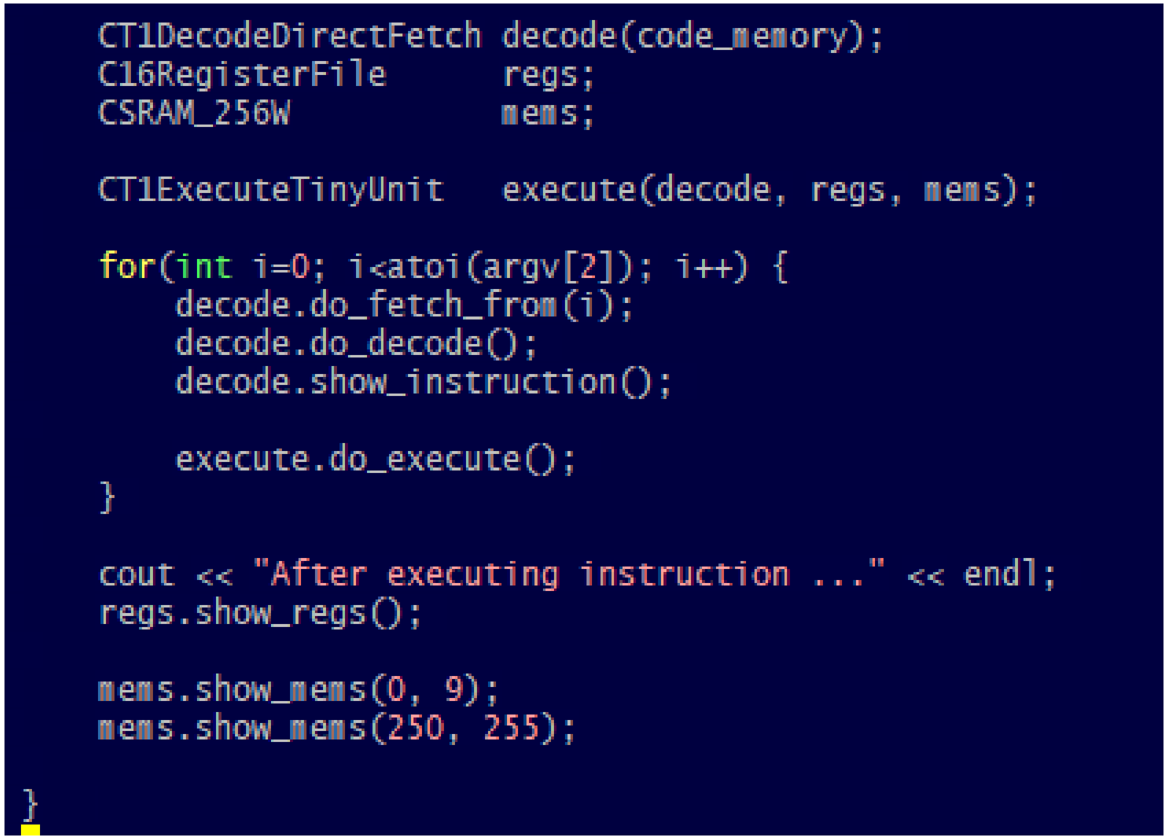


```
0011000000000001
0011000100000001
0011001000000010
0011001100000000
0100001100100000
0100001100100000
0101001100010000
0001001100000001
0000011100000001
0001001000000100
0000100000000100
0011111000000111
0011111100000001
0100111111100000
0011110111111111
0011110011111110
0001110011111111
0001110111111110
0000101111111110
```

ACES Lab.

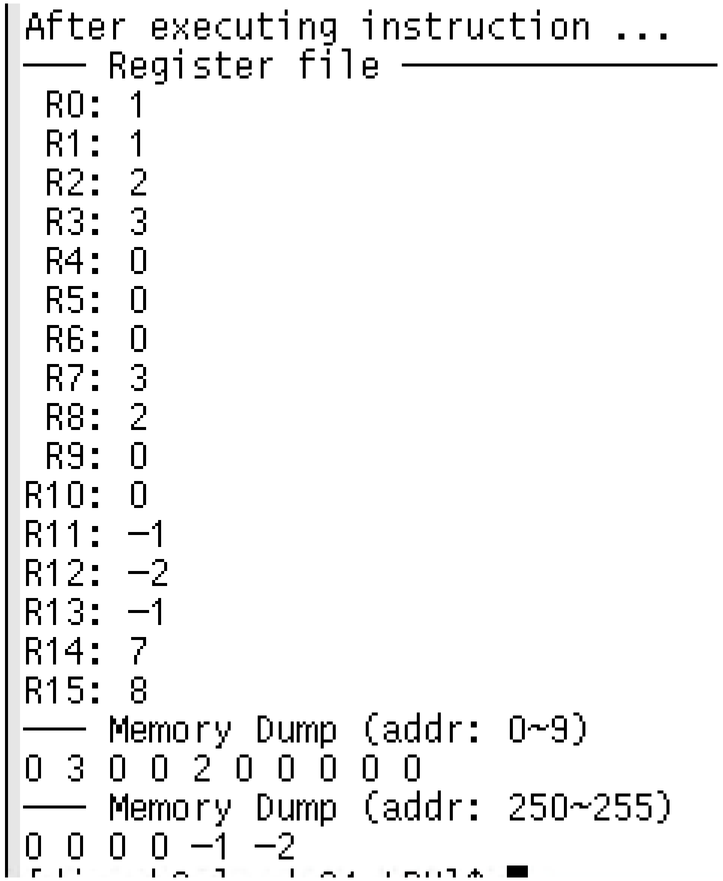
TPU 실행

main.cpp



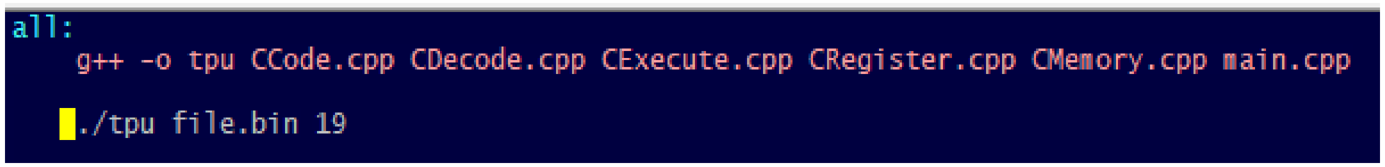
```
    CT1DecodeDirectFetch decode(code_memory);
    C16RegisterFile      regs;
    CSRAM_256W           mems;

    CT1ExecuteTinyUnit   execute(decode, regs, mems);

    for(int i=0; i<atoi(argv[2]); i++) {
        decode.do_fetch_from(i);
        decode.do_decode();
        decode.show_instruction();

        execute.do_execute();
    }

    cout << "After executing instruction ..." << endl;
    regs.show_regs();

    mems.show_mems(0, 9);
    mems.show_mems(250, 255);

}
```

실행결과



```
After executing instruction ...
--- Register file ----------
 R0: 1
 R1: 1
 R2: 2
 R3: 3
 R4: 0
 R5: 0
 R6: 0
 R7: 3
 R8: 2
 R9: 0
R10: 0
R11: -1
R12: -2
R13: -1
R14: 7
R15: 8
--- Memory Dump (addr: 0~9)
0 3 0 0 2 0 0 0 0 0
--- Memory Dump (addr: 250~255)
0 0 0 0 -1 -2
```

Makefile



```
all:
	g++ -o tpu CCode.cpp CDecode.cpp CExecute.cpp CRegister.cpp CMemory.cpp main.cpp

	./tpu file.bin 19
```

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)