마이크로프로세서

- 명령어 추가 (ADD, SUB) -

Daejin Park

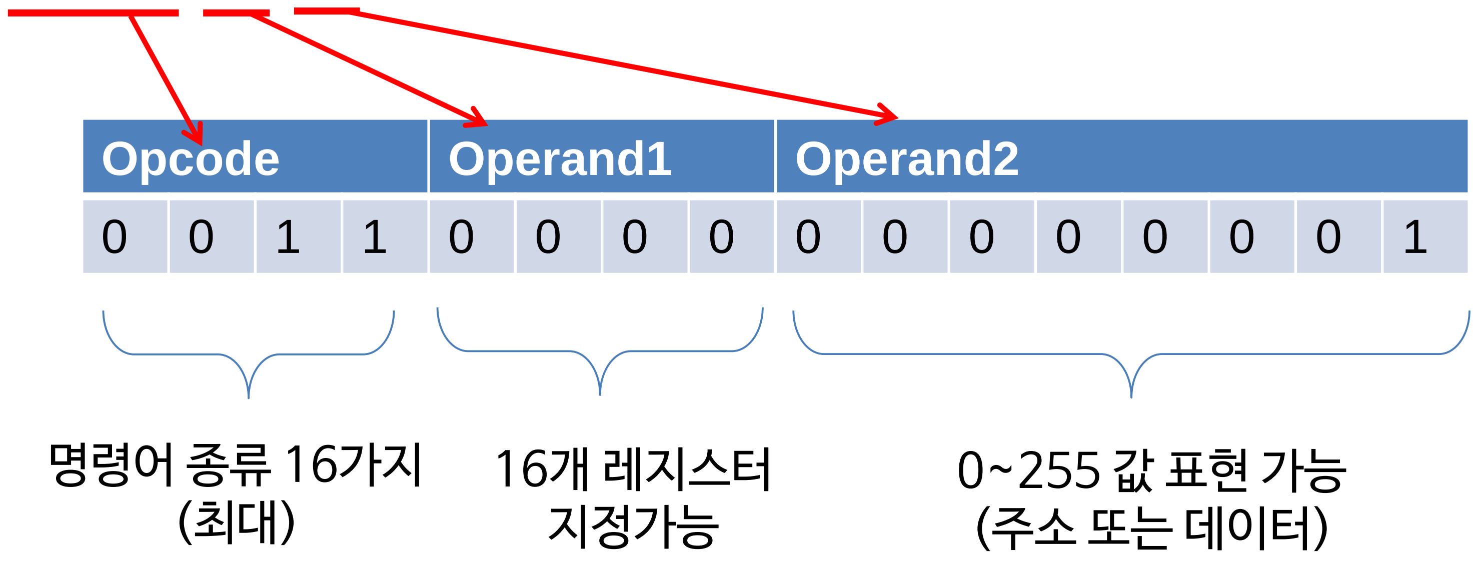
School of Electronics Engineering, KNU, KOREA

2019.04.03

ACES Lab.

tPU ISA (Instruction Set Architecture)

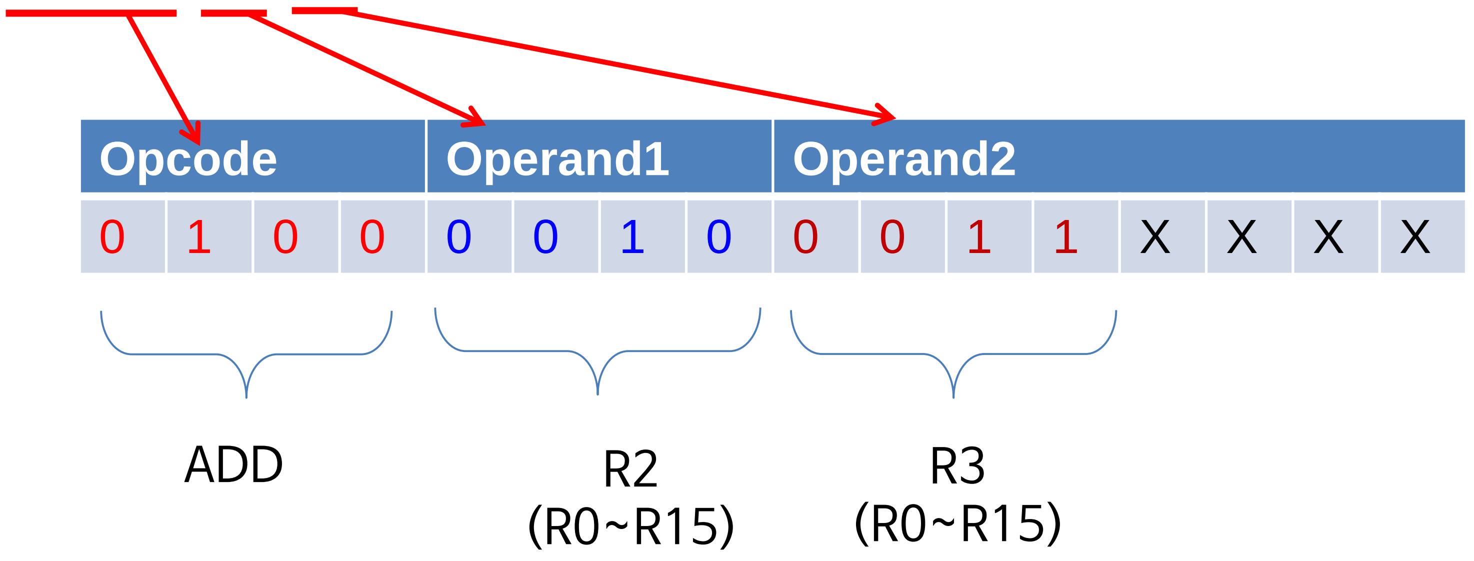
- MOV3 R0, #1



tPU ISA (Instruction Set Architecture)

- ADD Rn, Rm
  - Rn = Rn + Rm

- Ex. ADD R2, R3

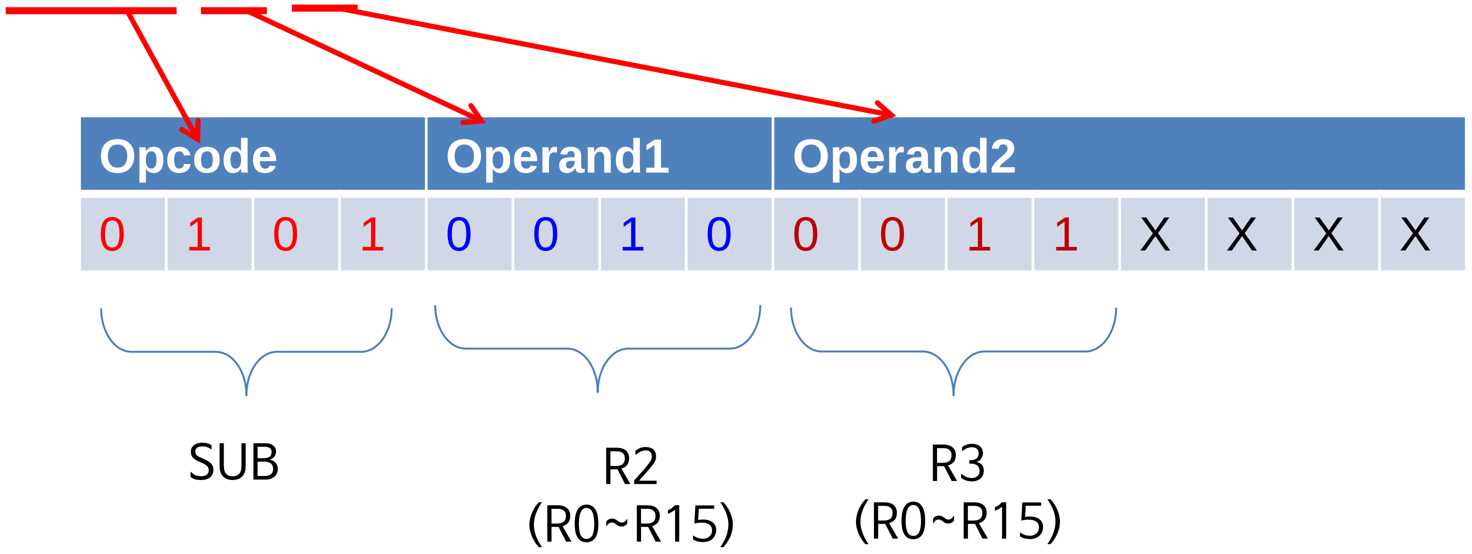


Opcode				Operand1				Operand2							
0	1	0	0	0	0	1	0	0	0	1	1	X	X	X	X

tPU ISA (Instruction Set Architecture)

- SUB Rn, Rm
  - Rn = Rn - Rm

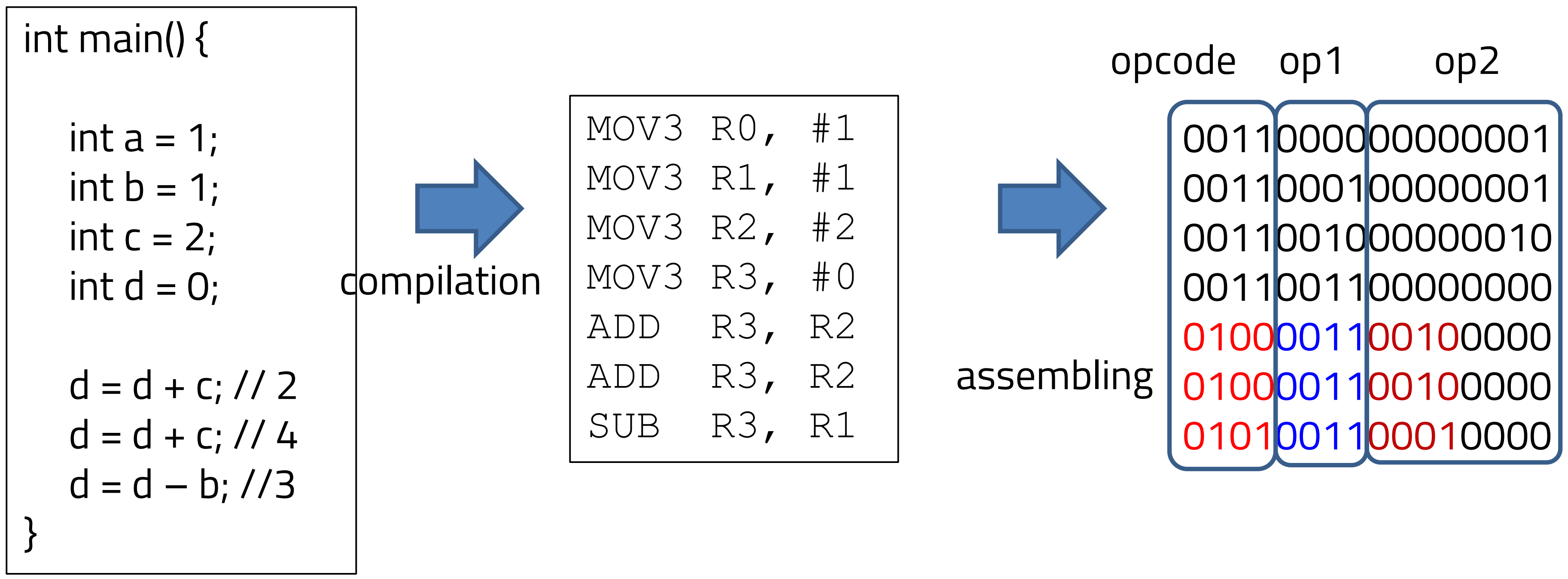
- Ex. SUB R2, R3



ACES Lab.

기계코드 변환

- 실습

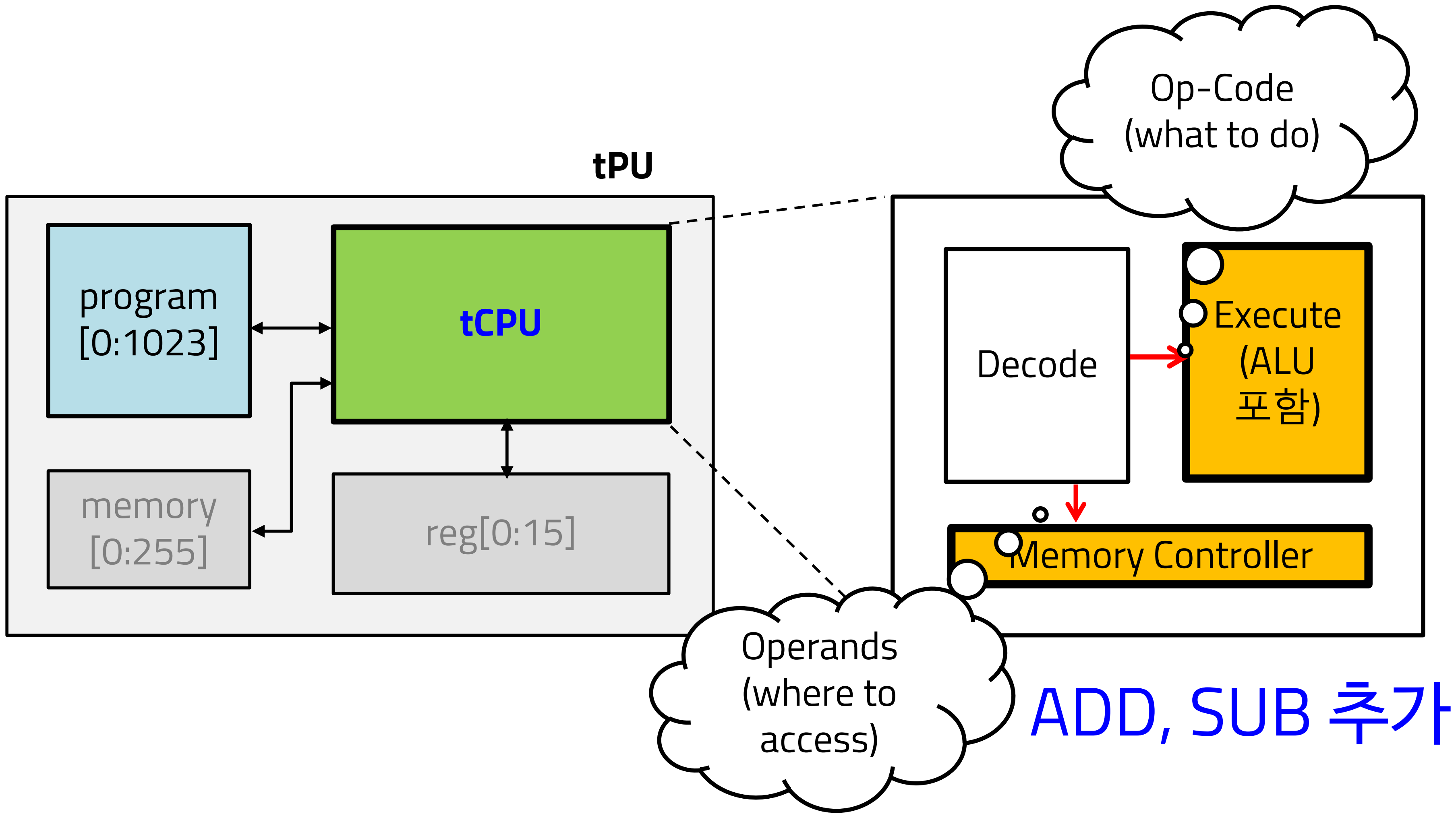


```
int main() {

    int a = 1;
    int b = 1;
    int c = 2;
    int d = 0;

    d = d + c; // 2
    d = d + c; // 4
    d = d – b; //3
}
```

```
MOV3  R0,  #1
MOV3  R1,  #1
MOV3  R2,  #2
MOV3  R3,  #0
ADD   R3,  R2
ADD   R3,  R2
SUB   R3,  R1
```

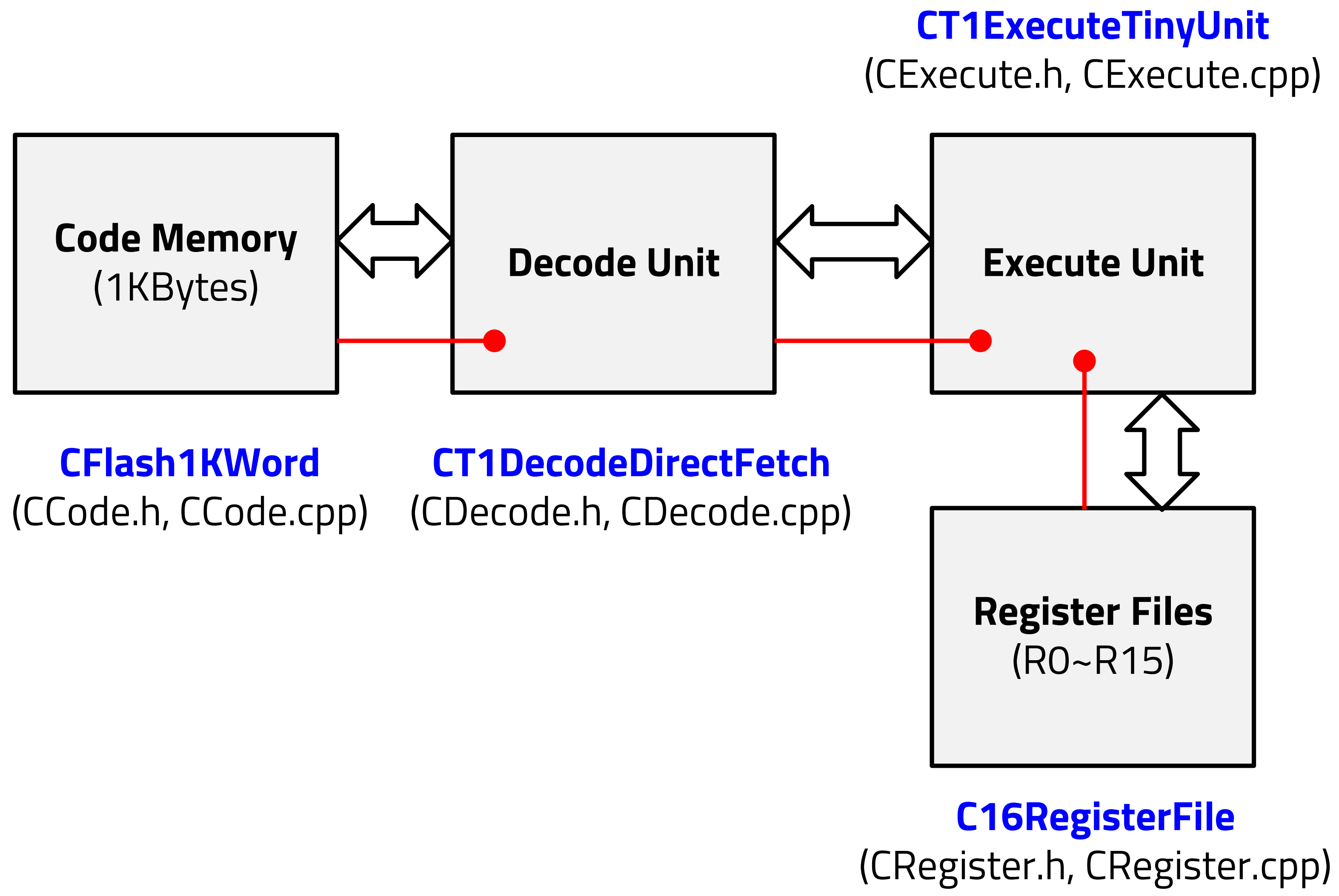
```
0011000000000001
0011000100000001
0011001000000010
0011001100000000
0100001100100000
0100001100100000
0101001100010000
```

Controlling Execute Unit and Memory



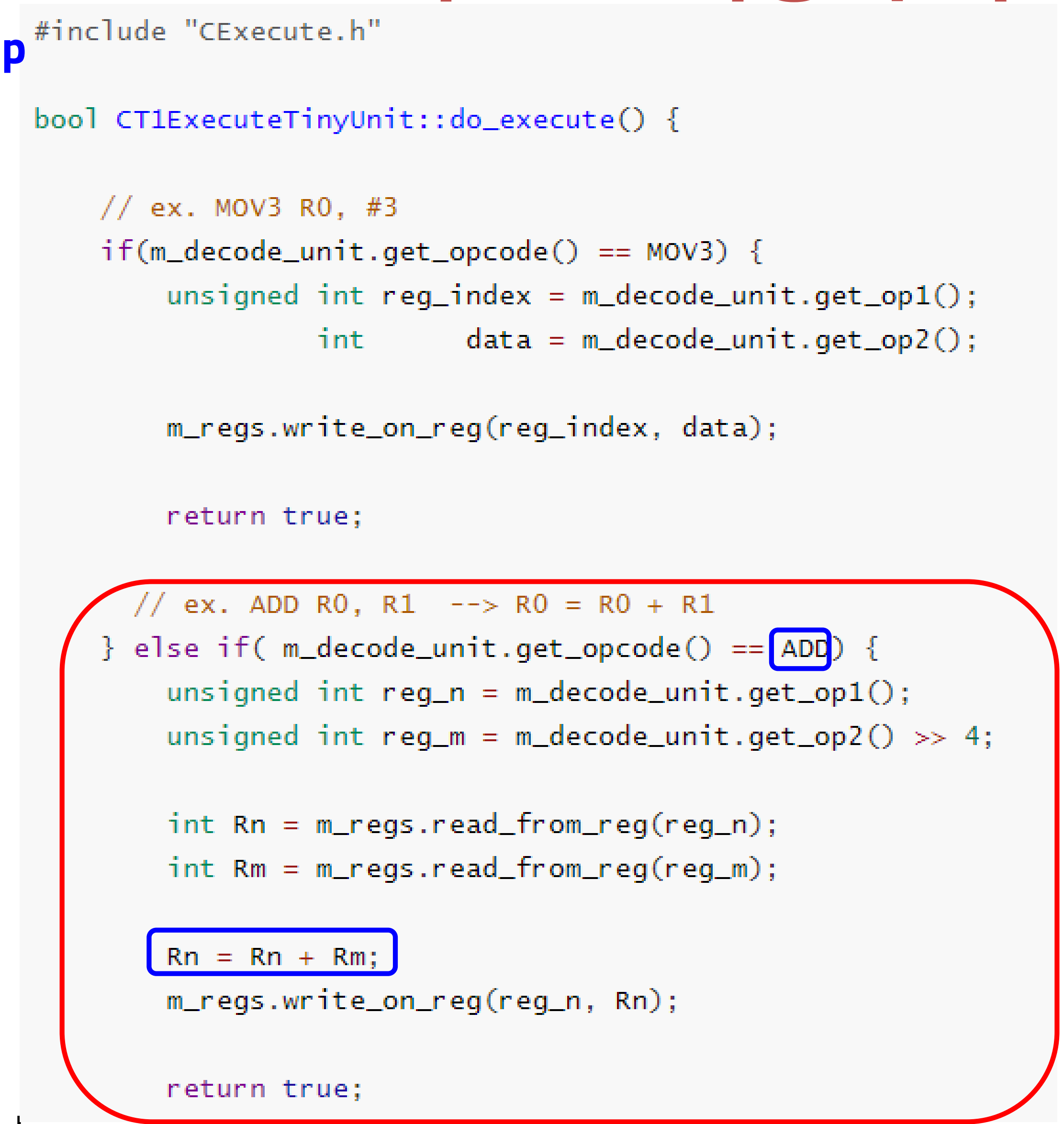
ADD, SUB 추가

Overall Architecture



Execution Unit 에 ADD 기능 추가

CExecute.cpp



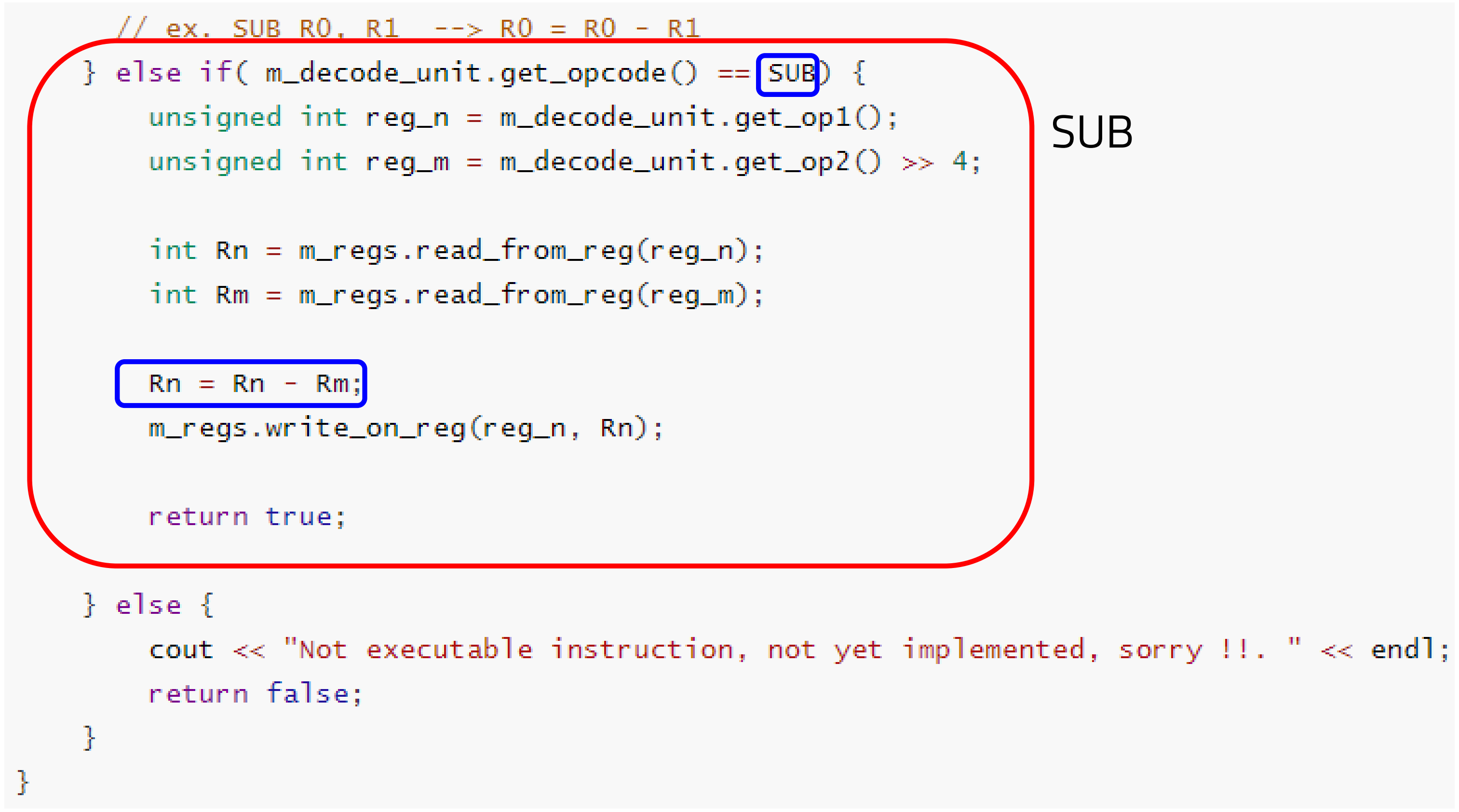
```
#include "CExecute.h"

bool 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);

        return true;

      // 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;

        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;
```

ADD

다음페이지 계속

Execution Unit 에 ADD 기능 추가

CExecute.cpp 앞 페이지 내용 연이어..



```
      // 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;

        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;
    }
}
```

SUB

Decode 일부수정

CExecute.h

```cpp
#include <iostream>
#include "CDecode.h"
#include "CRegister.h"

#pragma once

using namespace std;

class CExecute {
public:
    CExecute() { }
    virtual ~CExecute() { }

};

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

    bool do_execute();
private:

    CT1DecodeDirectFetch& m_decode_unit;
    C16RegisterFile&      m_regs;


};
```

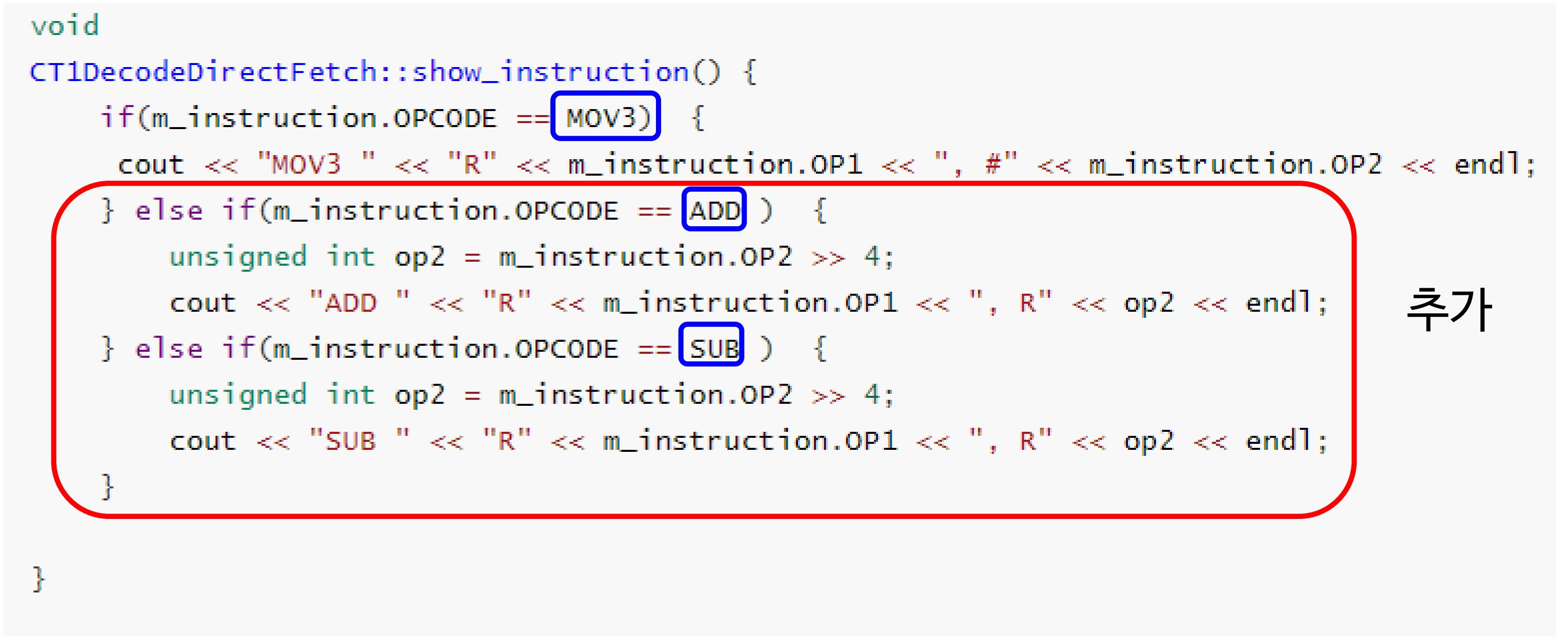
이동

CDecode.h

```cpp
#include <iostream>
#include "CCode.h"

#pragma once

using namespace std;

class CDecode {
public:
    CDecode() { }
    virtual ~CDecode() { }

};

enum {  R0=0,R1,   R2,  R3,
        R4,  R5,   R6,  R7,
        R8,  R9,  R10, R11,
       R12, R13,  R14, R15 } ;

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


typedef struct {
    unsigned int OPCODE  : 4;
    unsigned int OP1     : 4;
             int OP2     : 8;
} SInstruction;


class CT1DecodeDirectFetch : public CDecode {
public:
    CT1DecodeDirectFetch(CFlash1KWord& code) : m_code_memory(code) { }
    virtual ~CT1DecodeDirectFetch() { }



....
```

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;
        cout << "ADD " << "R" << m_instruction.OP1 << ", R" << op2 << endl;
    } else if(m_instruction.OPCODE == SUB )  {
        unsigned int op2 = m_instruction.OP2 >> 4;
        cout << "SUB " << "R" << m_instruction.OP1 << ", R" << op2 << endl;
    }

}
```

추가

ACES Lab.

file.bin (기계코드)

```
0011000000000001
0011000100000001
0011001000000010
0011001100000000
```

file.bin

```
0011000000000001
0011000100000001
0011001000000010
0011001100000000
0100001100100000
0100001100100000
0101001100010000
```

```
MOV3 R0, #1
MOV3 R1, #1
MOV3 R2, #2
MOV3 R3, #0
```

```
MOV3 R0, #1
MOV3 R1, #1
MOV3 R2, #2
MOV3 R3, #0
ADD  R3, R2
ADD  R3, R2
SUB  R3, R1
```

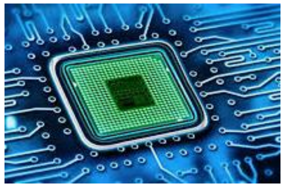
Makefile

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


  ./tpu file.bin 7
```

프로세서 (HW)

펌웨어 (SW)

코드실행 라인수

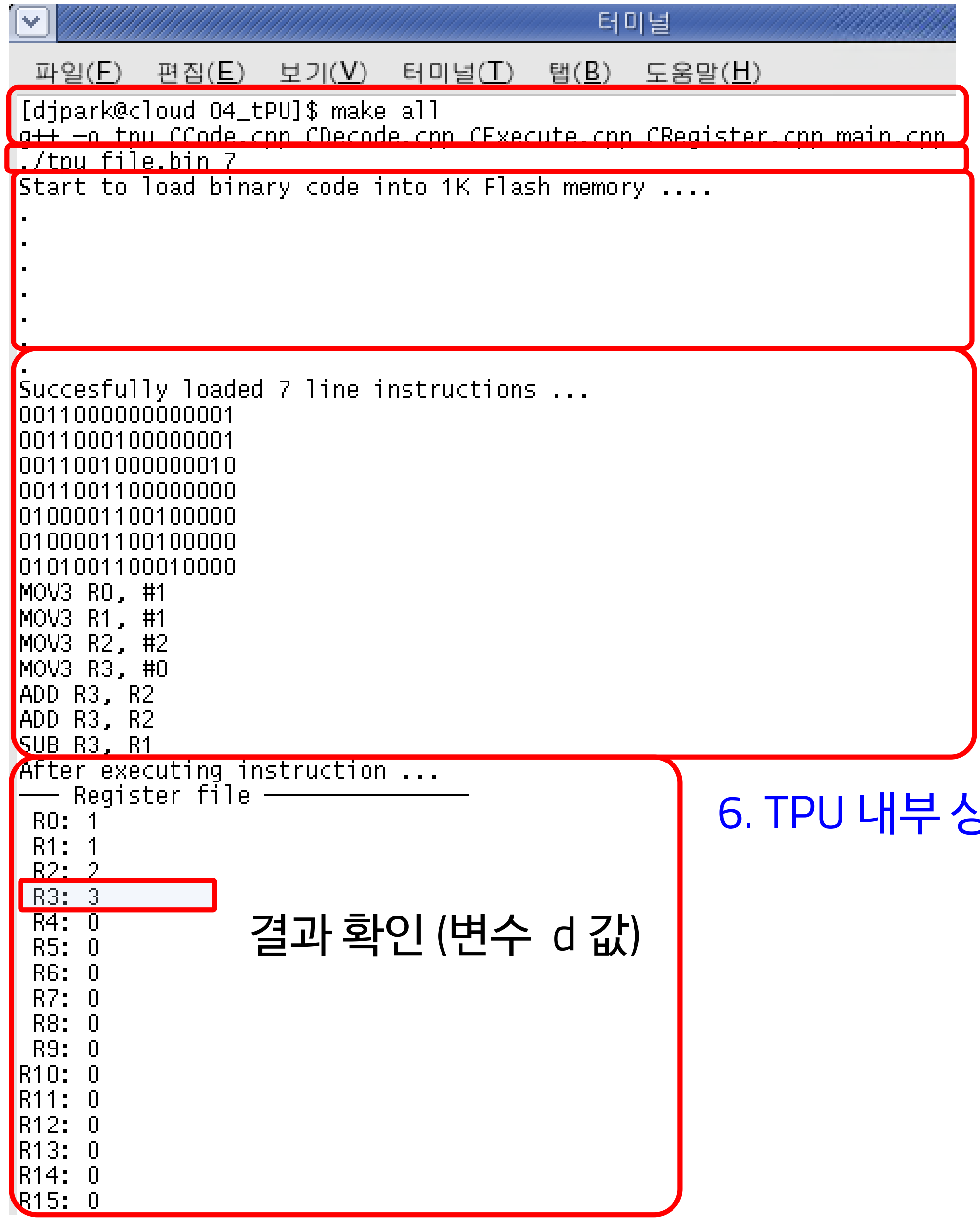


file.bin

```
0011000000000001
0011000100000001
0011001000000010
0011001100000000
0100001100100000
0100001100100000
0101001100010000
```

ACES Lab.

TPU 프로세서 코드 실행 결과



```
[djpark@cloud 04_tPU]$ make all
g++ -o tpu CCode.cpp CDecode.cpp CExecute.cpp CRegister.cpp main.cpp
./tpu file.bin 7
Start to load binary code into 1K Flash memory ....
.
.
.
.
.
.
.
Succesfully loaded 7 line instructions ...
0011000000000001
0011000100000001
0011001000000010
0011001100000000
0100001100100000
0100001100100000
0101001100010000
MOV3 R0, #1
MOV3 R1, #1
MOV3 R2, #2
MOV3 R3, #0
ADD R3, R2
ADD R3, R2
SUB R3, R1
After executing instruction ...
--- Register file ---------------
 R0: 1
 R1: 1
 R2: 2
 R3: 3
 R4: 0
 R5: 0
 R6: 0
 R7: 0
 R8: 0
 R9: 0
R10: 0
R11: 0
R12: 0
R13: 0
R14: 0
R15: 0
```

1. 컴파일
2. 실행
3. 코드 loading

4. Decoding,
5. Execution

6. TPU 내부 상태 확인

결과 확인 (변수 d 값)

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)