

ISS Modeling

Negin Safari

Supervisors : Dr.Navabi & Katayoon Basharkhah

Outline

Section1

Introduction to RL78-S2

- Memory Space
- Registers
- Addressing
- Instructions

Section2

Implementation Methodology

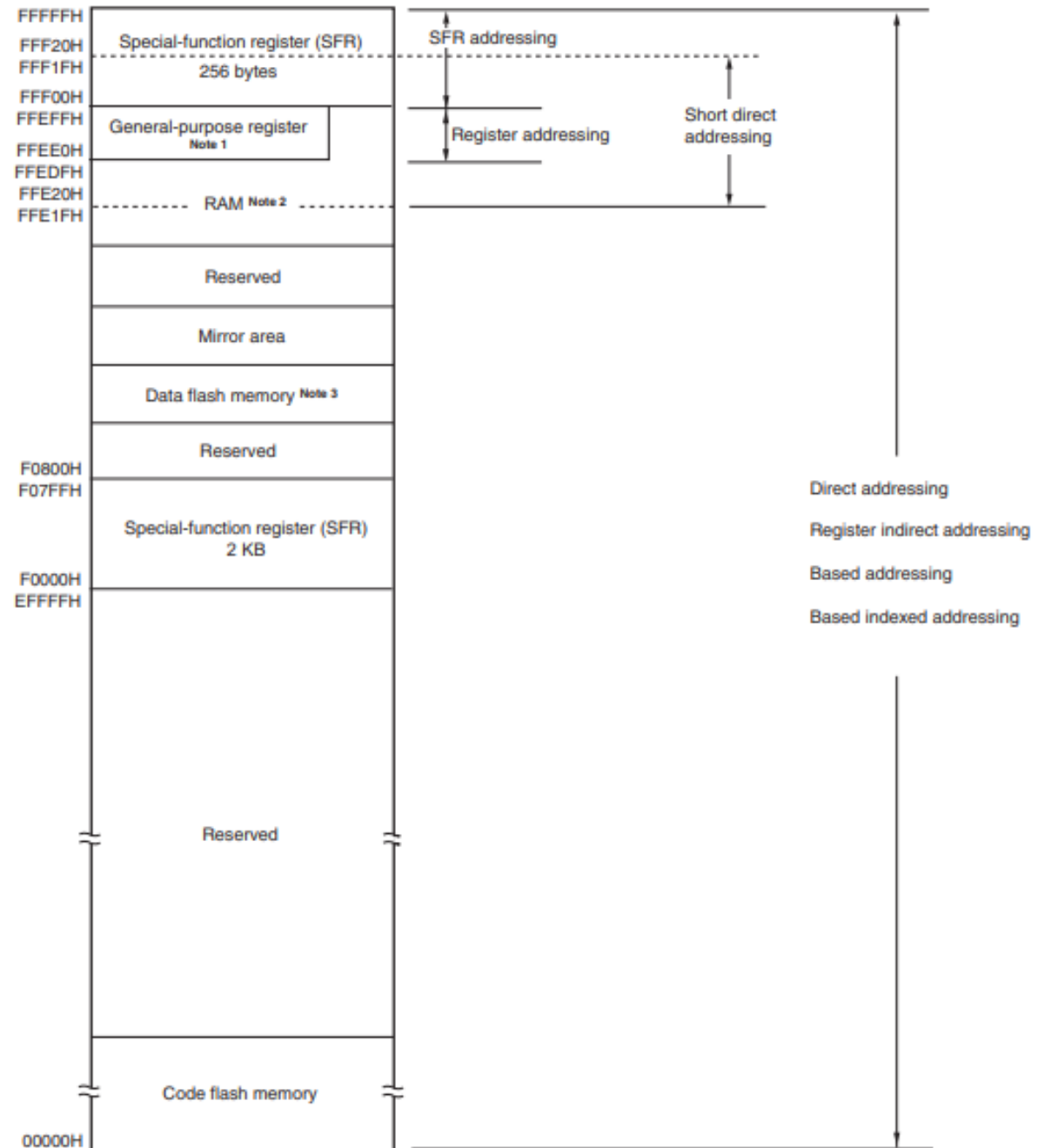
- First Methodology
- Second Methodology

RL78-S2


RENESAS

Memory Space

- 8 bit memory with 2^{20} rows.
- This memory is separated into several parts. Each part is used for different purposes.



Registers

- Control Registers 
 - Program Counter (PC)
 - Program Status Word (PSW)
 - Stack Pointer (SP)
- General-Purpose Registers → - 4 register banks
- ES and CS Registers
- Special Function Registers (SFRs)

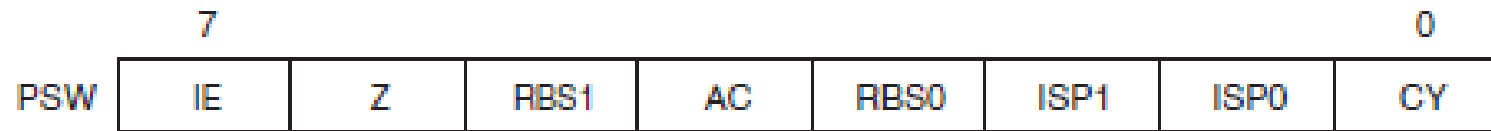
Program Counter (PC)

- It is a 20 bit register that holds the address information of the next program to be executed.
- In normal operation, PC is automatically incremented according to the **number of bytes** of the instruction.

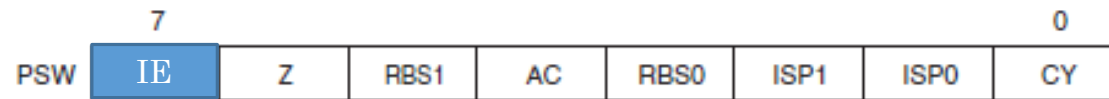


Program Status Word (PSW)

- It is an 8 bit register located at `0xFFFFFA` in the memory.

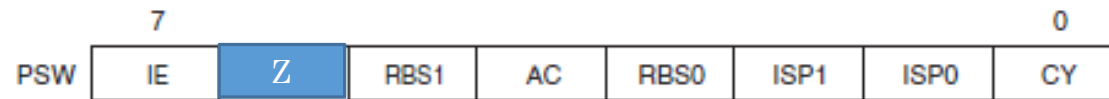


PSW - IE



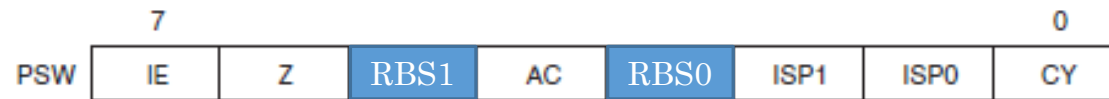
This flag controls the interrupt request acknowledgement operations of the CPU.

PSW - Z



When the operation or comparison result is zero or equal, this flag is set (1). It is reset (0) in all other cases

PSW - RBS



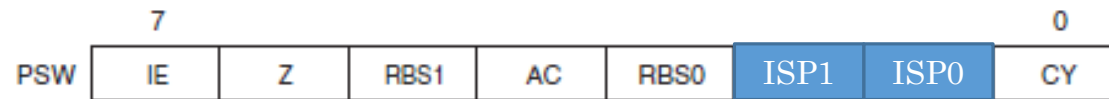
These are 2-bit flags used to select one of the four register banks.

PSW - AC



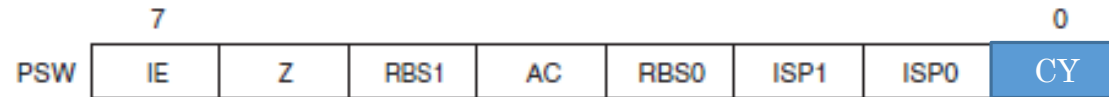
If the operation result has a carry from bit 3 or a borrow at bit 3, this flag is set (1). It is reset (0) in all other cases.

PSW - ISP



This flag manages the priority of acknowledgeable maskable vectored interrupts.

PSW - CY



This flag stores an overflow or underflow upon add/subtract instruction execution.

Stack Pointer(SP)

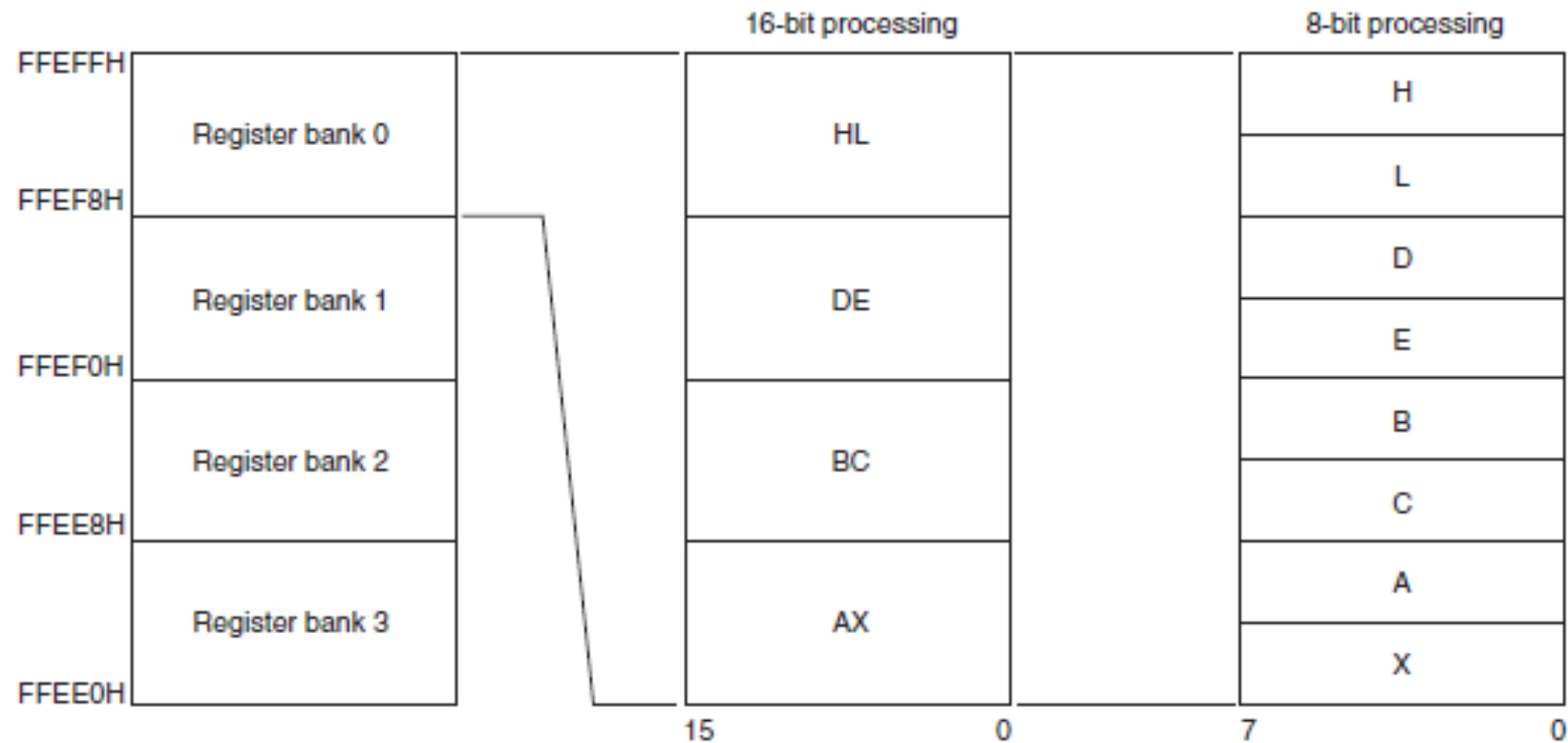
- This is a 16-bit register that holds the start address of the memory stack area.
- It is located at 0xFFFF9 and 0xFFFF8 in the memory.



4 register banks

- General-purpose registers are mapped at particular addresses (FFEE0H to FFEFFH) of the data memory.
- These registers consist of 4 banks, each bank consisting of eight 8-bit registers (X, A, C, B, E, D, L and H).
- Each register can be used as an 8-bit register, and two 8-bit registers can also be used in a pair as a 16-bit register (AX, BC, DE, and HL).

Register banks memory map



ES and CS registers

- The ES register and CS register are used to specify the higher address for data access and when a branch instruction is executed (register direct addressing), respectively.
- ES is located at 0xFFFFD in the memory.
- CS is located at 0xFFFFC in the memory.

	7	6	5	4	3	2	1	0
ES	0	0	0	0	ES3	ES2	ES1	ES0

	7	6	5	4	3	2	1	0
CS	0	0	0	0	CS3	CS2	CS1	CS0

Addressing

- Instruction Address Addressing
- Addressing for Processing Data Addresses

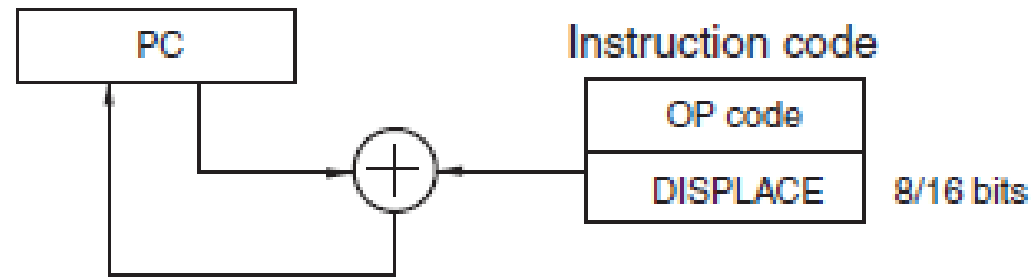
Instruction Address Addressing

- Relative addressing
- Immediate addressing
- Table indirect addressing
- Register direct addressing

Relative addressing

- Relative addressing:

Stores in the program counter (PC) the result of adding a displacement value included in the instruction word to the program counter (PC)'s value.



Immediate addressing

- Stores immediate data of the instruction word in the program counter
- Specifies the program address to be used as the branch destination.

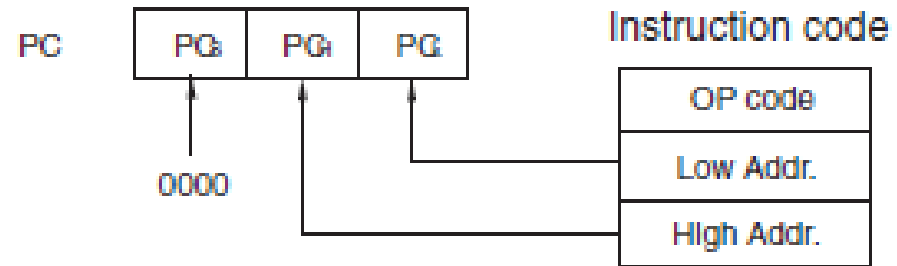
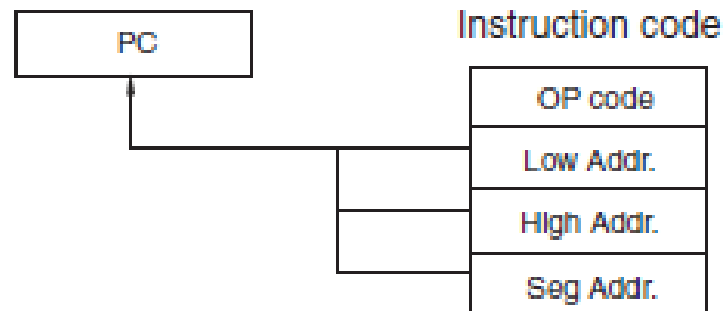
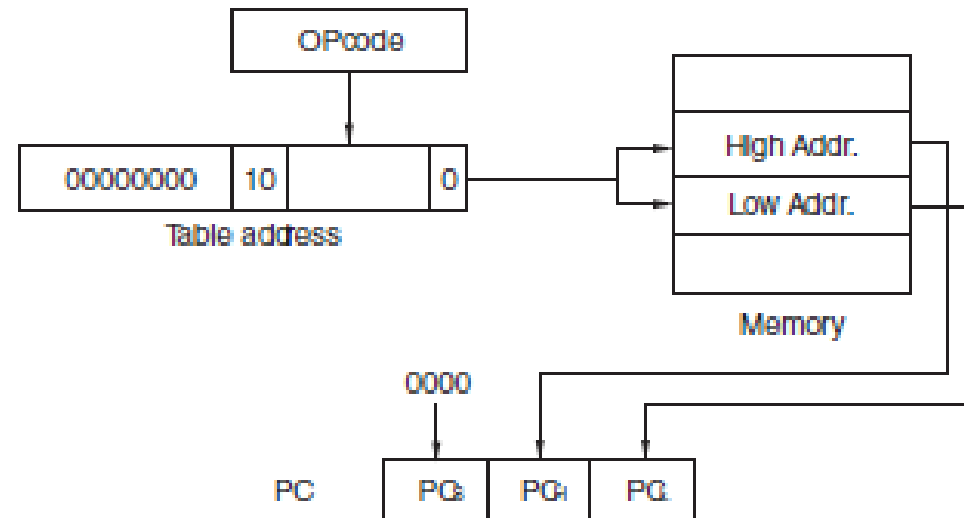


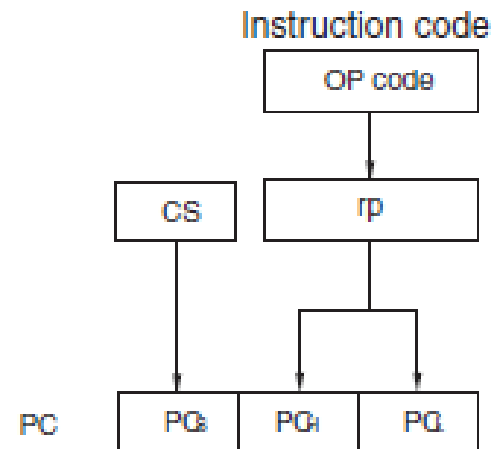
Table indirect addressing

- Specifies a table address in the CALLT table area (0080H to 00BFH) with the 5-bit immediate data in the instruction word
- Stores the contents at that table address and the next address in the program counter (PC) as 16-bit data
- Specifies the program address.



Register direct addressing

- stores in the program counter (PC) the contents of a general-purpose register pair (AX/BC/DE/HL).
- CS register of the current register bank specified with the instruction word as 20-bit data, and specifies the program address.

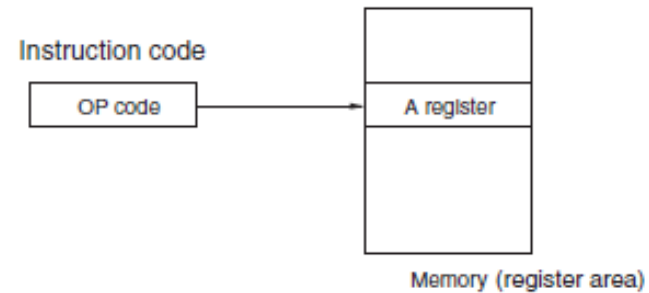


Addressing for Processing Data Addresses

- Implied addressing
- Register addressing
- Direct addressing
- Short direct addressing
- SFR addressing
- Register indirect addressing
- Based addressing
- Based indexed addressing
- Stack addressing

Implied addressing

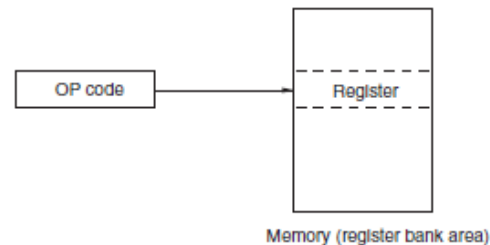
Instructions for accessing registers (such as accumulators) that have special functions are directly specified with the instruction word, without using any register specification field in the instruction word.



Register addressing

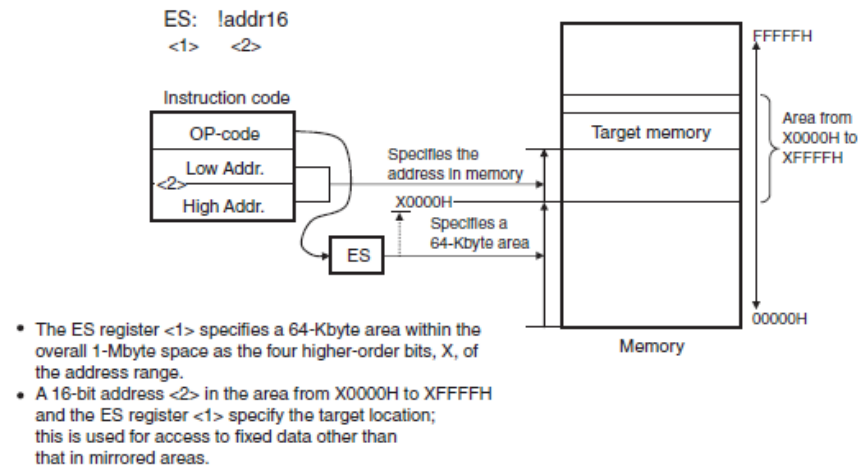
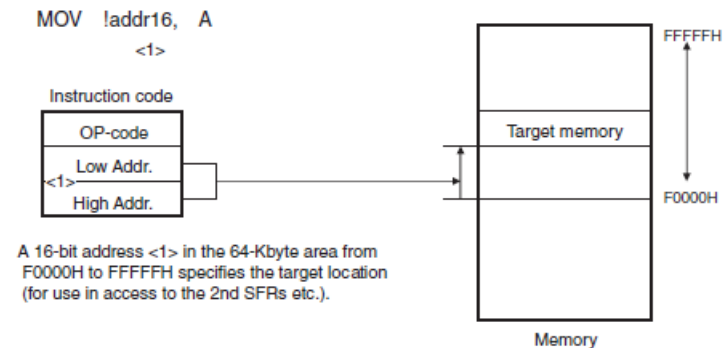
- Register addressing accesses a general-purpose register as an operand.
- The instruction word of 3-bit long is used to select an 8-bit register and the instruction word of 2-bit long is used to select a 16-bit register.

Identifier	Description
r	X, A, C, B, E, D, L, H
rp	AX, BC, DE, HL



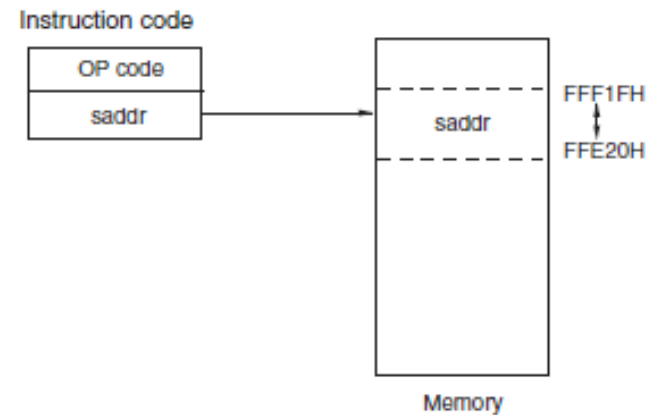
Direct addressing

Direct addressing uses immediate data in the instruction word as an operand address to directly specify the target address.



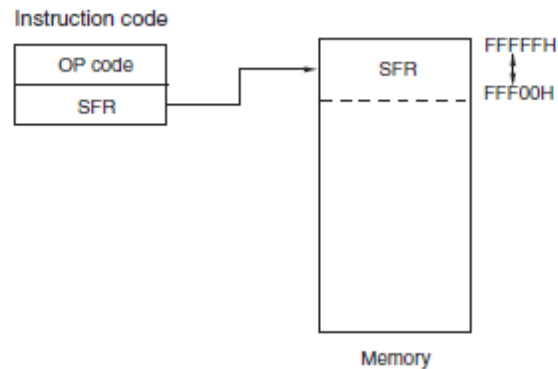
Short direct addressing

- Short direct addressing directly specifies the target addresses using 8-bit data in the instruction word.
- This type of addressing is applied only to the space from FFE20H to FFF1FH.



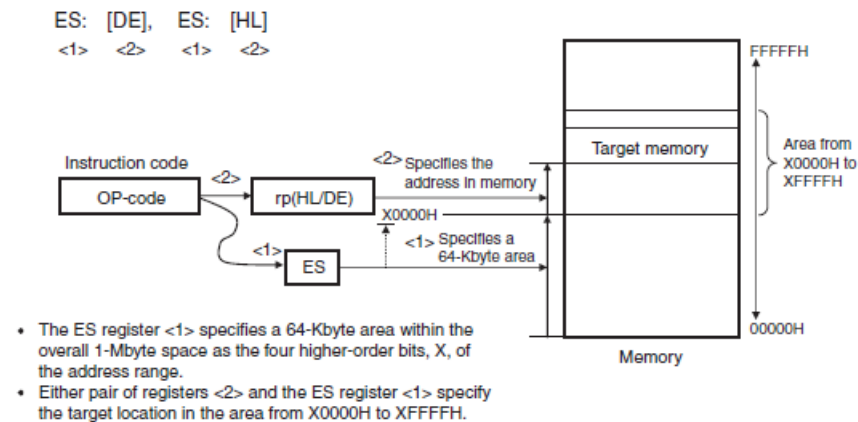
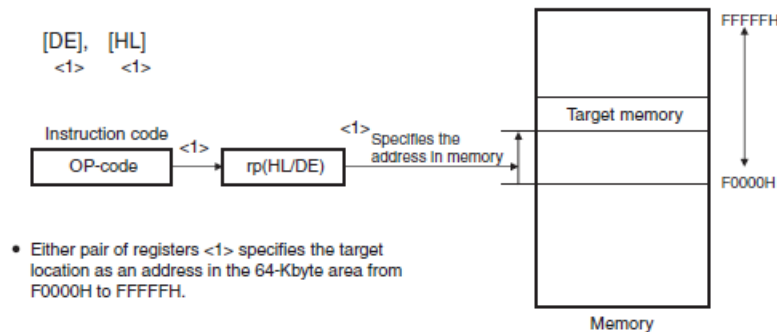
SFR addressing

- SFR addressing directly specifies the target SFR addresses using 8-bit data in the instruction word.
- This type of addressing is applied only to the space from FFF00H to FFFFFH.



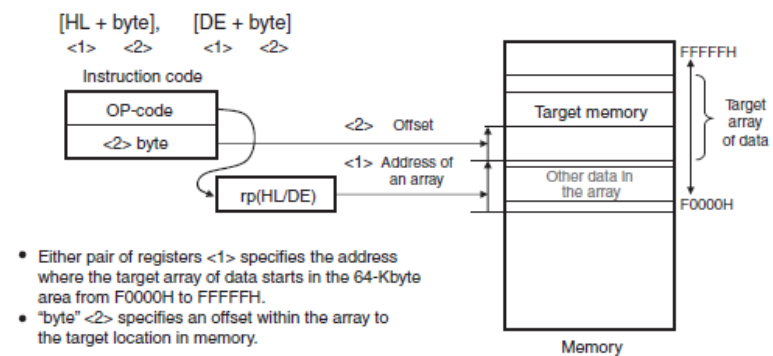
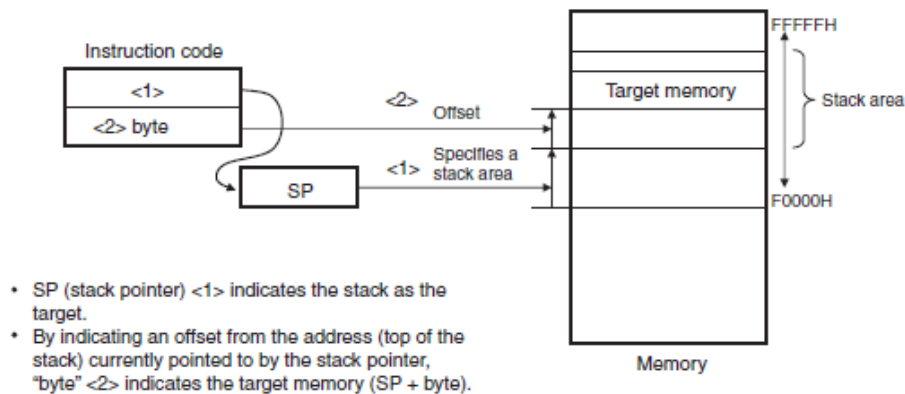
Register indirect addressing

Register indirect addressing directly specifies the target addresses using the contents of the register pair specified with the instruction word as an operand address.



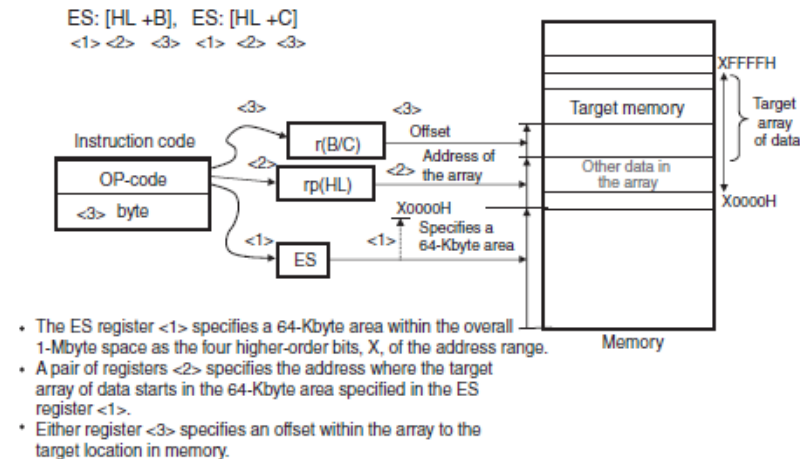
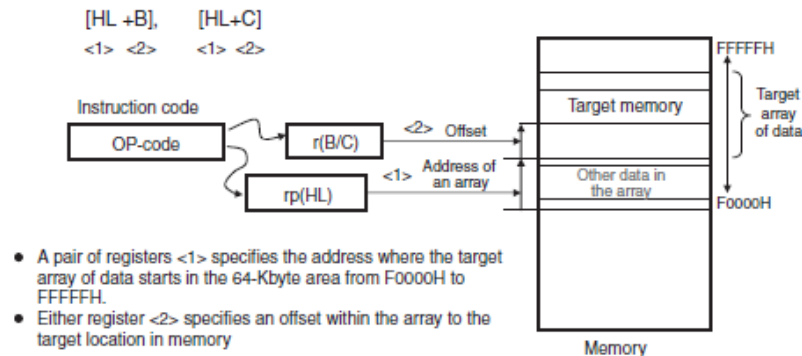
Based addressing

- It uses the contents of a register pair specified with the instruction word or 16-bit immediate data as a base address, and 8-bit immediate data or 16-bit immediate data as offset data.
- The sum of these values is used to specify the target address.



Based indexed addressing

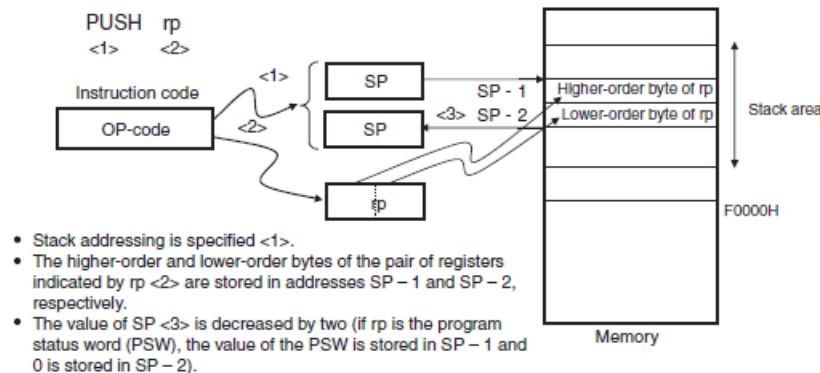
- Based indexed addressing uses the contents of a register pair specified with the instruction word as the base address, and the content of the B or C register similarly specified with the instruction word as offset address.
- The sum of these values is used to specify the target address.



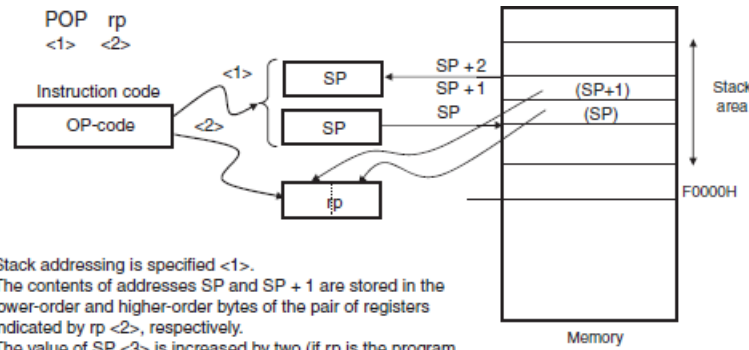
Stack addressing

- The stack area is indirectly addressed with the stack pointer (SP) contents.
- This addressing is automatically employed when the PUSH, POP, subroutine call, and return instructions are executed or the register is saved/restored upon generation of an interrupt request.

PUSH

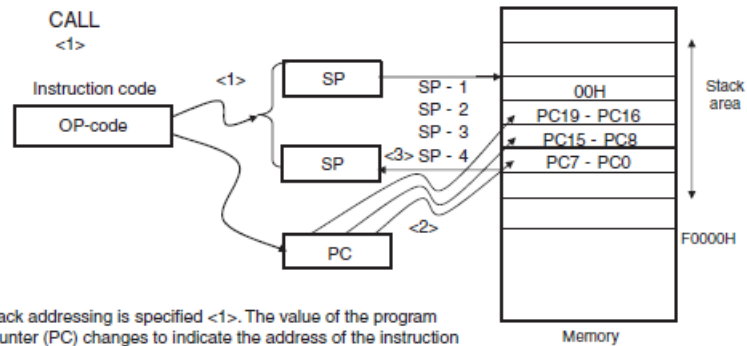


POP



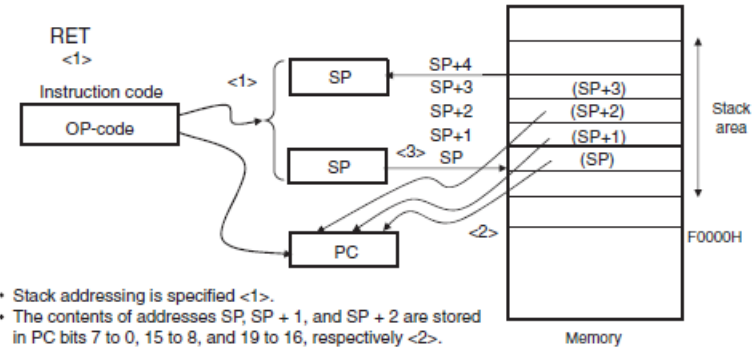
- Stack addressing is specified $\langle 1 \rangle$.
- The contents of addresses SP and SP + 1 are stored in the lower-order and higher-order bytes of the pair of registers indicated by rp $\langle 2 \rangle$, respectively.
- The value of SP $\langle 3 \rangle$ is increased by two (if rp is the program status word (PSW), the content of address SP + 1 is stored in the PSW).

CALL



- Stack addressing is specified $\langle 1 \rangle$. The value of the program counter (PC) changes to indicate the address of the instruction following the CALL instruction.
- 00H, the values of PC bits 19 to 16, 15 to 8, and 7 to 0 are stored in addresses SP - 1, SP - 2, SP - 3, and SP - 4, respectively $\langle 2 \rangle$.
- The value of the SP $\langle 3 \rangle$ is decreased by 4.

RET



- Stack addressing is specified $\langle 1 \rangle$.
- The contents of addresses SP, SP + 1, and SP + 2 are stored in PC bits 7 to 0, 15 to 8, and 19 to 16, respectively $\langle 2 \rangle$.
- The value of SP $\langle 3 \rangle$ is increased by four.

Instructions

This processor has 4 instruction maps.

Map I

The first byte indicates
the operation.
+ Prefix Instructions

Map II

The first byte is **71**
The **second** byte
indicates the operation.
+ Prefix Instructions

Map III

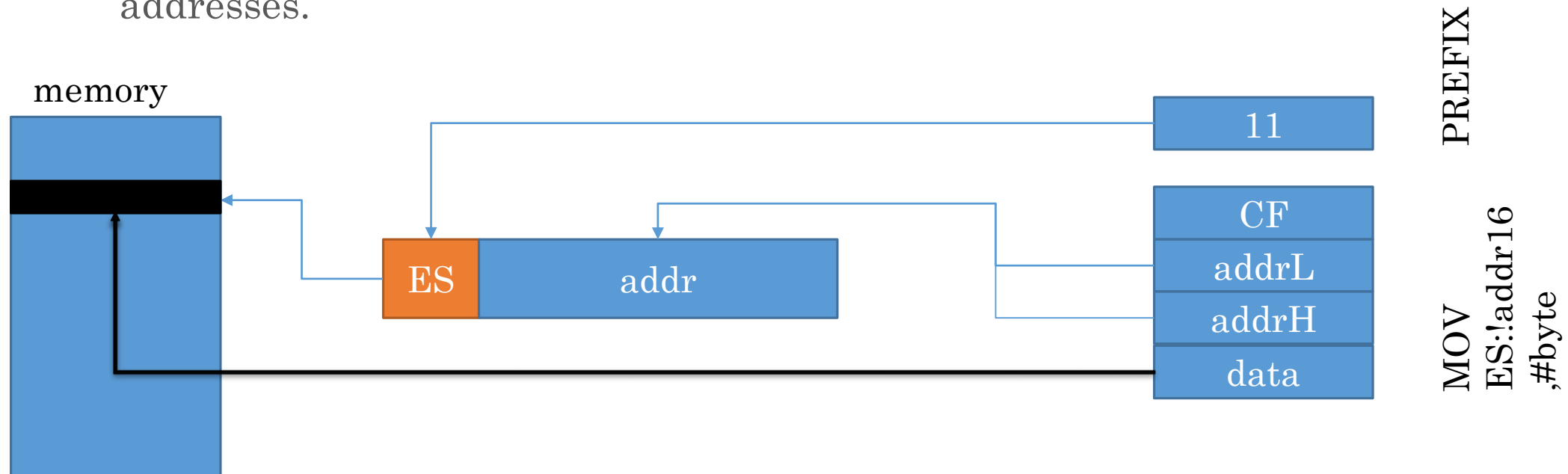
The first byte is **61**
The second byte
indicates the operation.
+ Prefix Instructions

Map IV

The first byte is **31**
The **second** byte
indicates the operation.
+ Prefix Instructions

Prefix Instructions

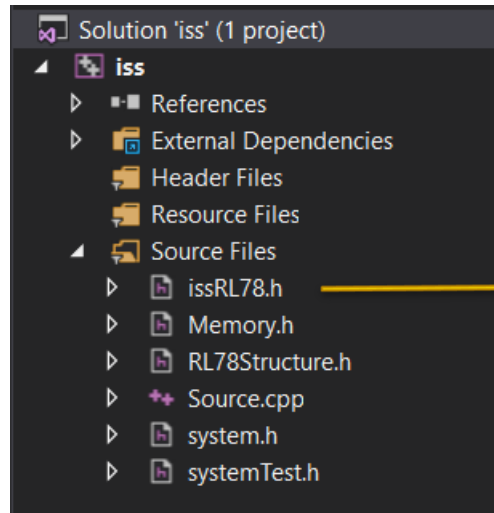
- These instructions are located in all maps.
- PREFIX is a 1 Byte instruction that takes place before the prefix instructions.
- This instruction enables the use of ES register as a prefix for addresses.



Methodology

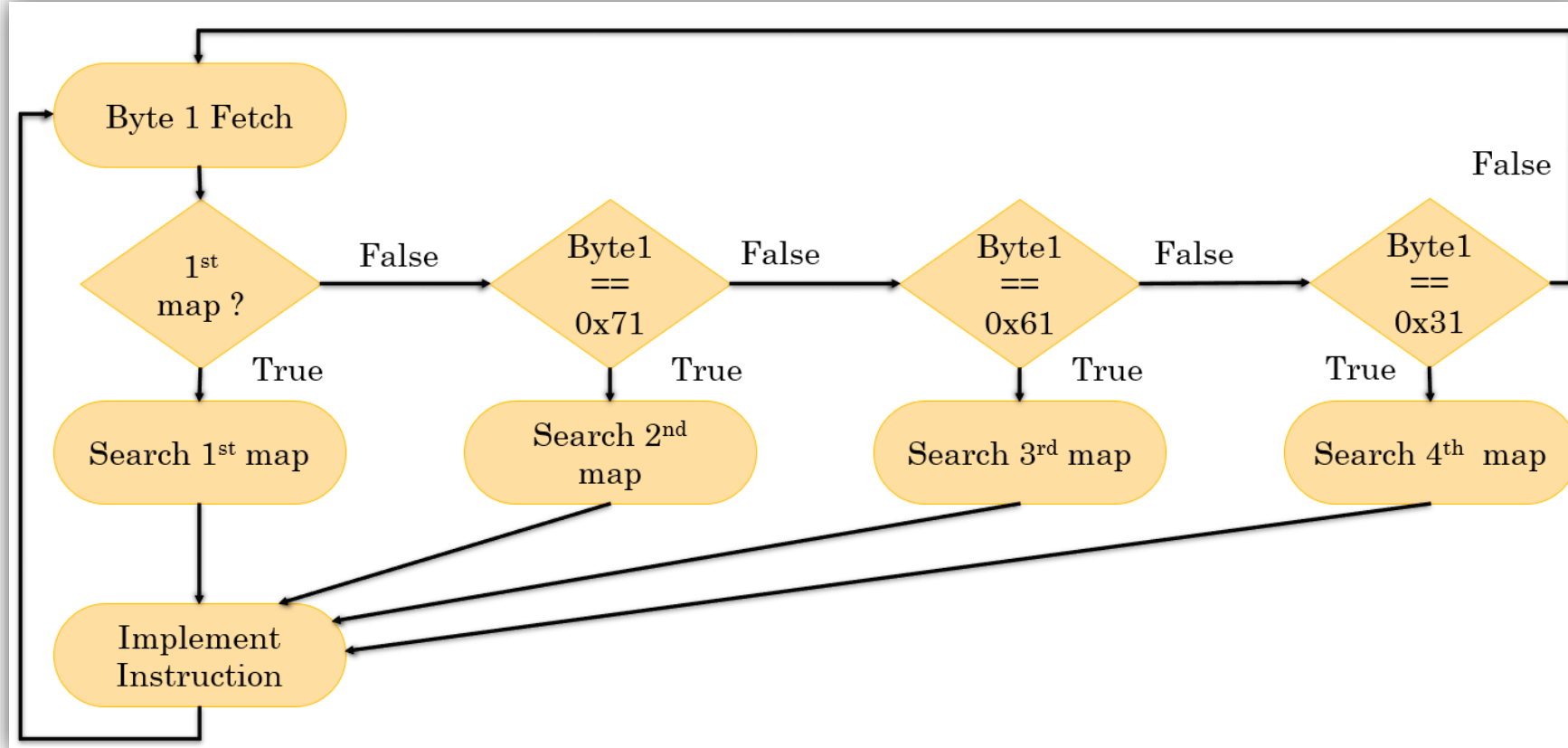
- For implementation we started with a simple system C code and after covering a good number of instructions we used a more configurable template based on OVP codes.

First methodology



```
SC_MODULE(issRL78) {  
    sc_in<sc_logic> clk;  
    sc_in<sc_logic> memReady;  
  
    sc_inout<sc_lv<8>> dataBus;  
  
    sc_out<sc_logic> readMem, writeMem;  
    sc_out<sc_lv<20>> addrBus;  
  
    struct RL78Struct ST;  
  
    SC_HAS_PROCESS(issRL78);  
  
    issRL78(sc_module_name) {  
  
        ST.adrSpace = int(pow(2, 20));  
  
        SC_THREAD(abstractSimulation);  
        sensitive << clk.pos();  
    }  
}
```

Flowchart



First methodology

```
while (true) {
```

```
    Byte1 = "00000000";  
    Byte2 = "00000000";  
    Byte3 = "00000000";  
    Byte4 = "00000000";
```

```
    writeMem = SC_LOGIC_0;  
    Byte1 = Fetch1Byte();
```

Fetching the first byte

```
    cout << "    ### PC = "
```

```
    bool SecondMapTrue = checkFirstMap(Byte1, Byte2, Byte3, Byte4);
```

```
    if (SecondMapTrue) {
```

```
        bool ThrdMapTrue = checkSndMap(Byte1, Byte2, Byte3, Byte4);
```

```
        if (ThrdMapTrue) {  
            cout << "N FOUND " << endl;
```

Checking other maps

If it was in first map it is executed and it will return false

Otherwise

```
        getchar();
```

```
bool issRL78::checkFirstMap(sc_lv <8>& Byte1,
```

```
    sc_lv<8> dataA_8b = 0;  
    sc_lv<8> dataB_8b = 0;  
    sc_lv<16> dataA_16b = 0;  
    sc_lv<16> dataB_16b = 0;  
    sc_lv<8> result_8b = 0;  
    sc_lv<16> result_16b = 0;  
    sc_lv<20> addrHL = 0;  
    sc_lv<8> psw = 0;
```

```
    switch (Byte1.to_uint())
```

```
    {  
        case _1_0C_ADD: { ...  
    }
```

```
        case _1_02_ADDW: { ...  
    }
```

```
        case _1_00_NOP: { ...  
    }
```

```
        case _1_04_ADDW: { ...  
    }
```

```
        case _1_06_ADDW: { ...
```


Functions used for First Methodology

Used for flags
Getting specific
addresses

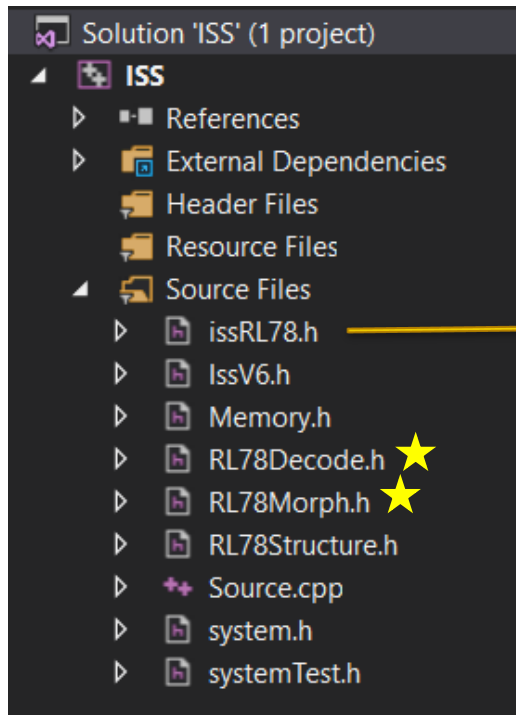
Used for stack
instructions

```
void issRL78::fillZ(int result_9b) { ...  
}  
void issRL78::fillCY(sc_lv<1> msb1, sc_lv<1> msb2, sc_lv<1> msb0, bool sub)  
{  
}  
void issRL78::fillAC(sc_logic op1, sc_logic op2, sc_logic result) { ...  
}  
sc_lv<20> issRL78::getRegAddr(int operand, sc_lv<2> Bank_Sel) { ...  
}  
sc_lv<8> issRL78::Fetch1Byte() { ...  
}  
sc_lv<8> issRL78::Read1Byte(sc_lv<20> addr) { ...  
}  
void issRL78::Write1Byte(sc_lv<8> data2write, sc_lv<20> addr) { ...  
}  
void issRL78::UpdatePSW() { ...  
}  
sc_lv<16> issRL78::ReadSP() { ...  
}  
void issRL78::UpdateSP(sc_lv<16> data) { ...  
}  
sc_lv<16> issRL78::Pop2ByteSP() { ...  
}  
void issRL78::Push2ByteSP(sc_lv<8> dh, sc_lv<8> dl) { ...  
}  
void issRL78::CallT(sc_lv<8> addr) { ...  
}  
sc_lv<20> issRL78::PrefixAddr() { ...
```

Pros and Cons of First Methodology

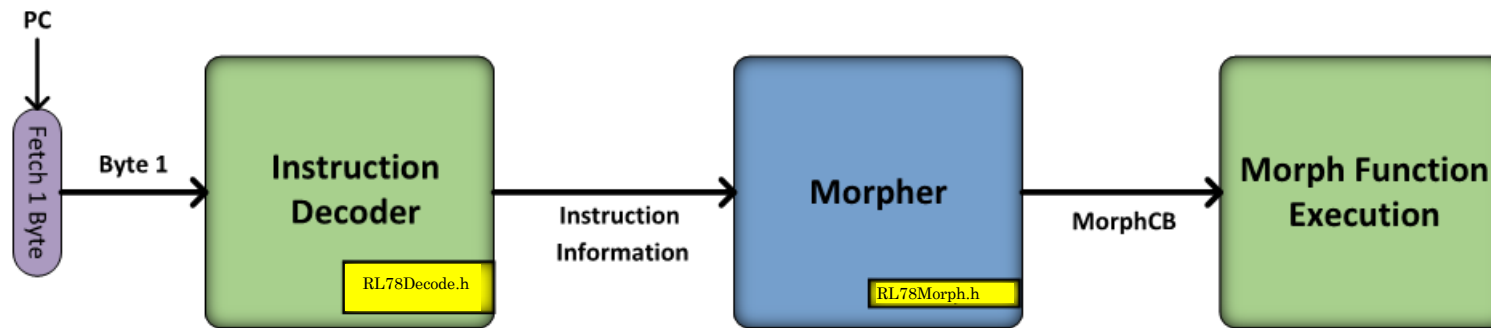
- Pros: easy to implement and fast
- Cons: difficult to understand

Second methodology

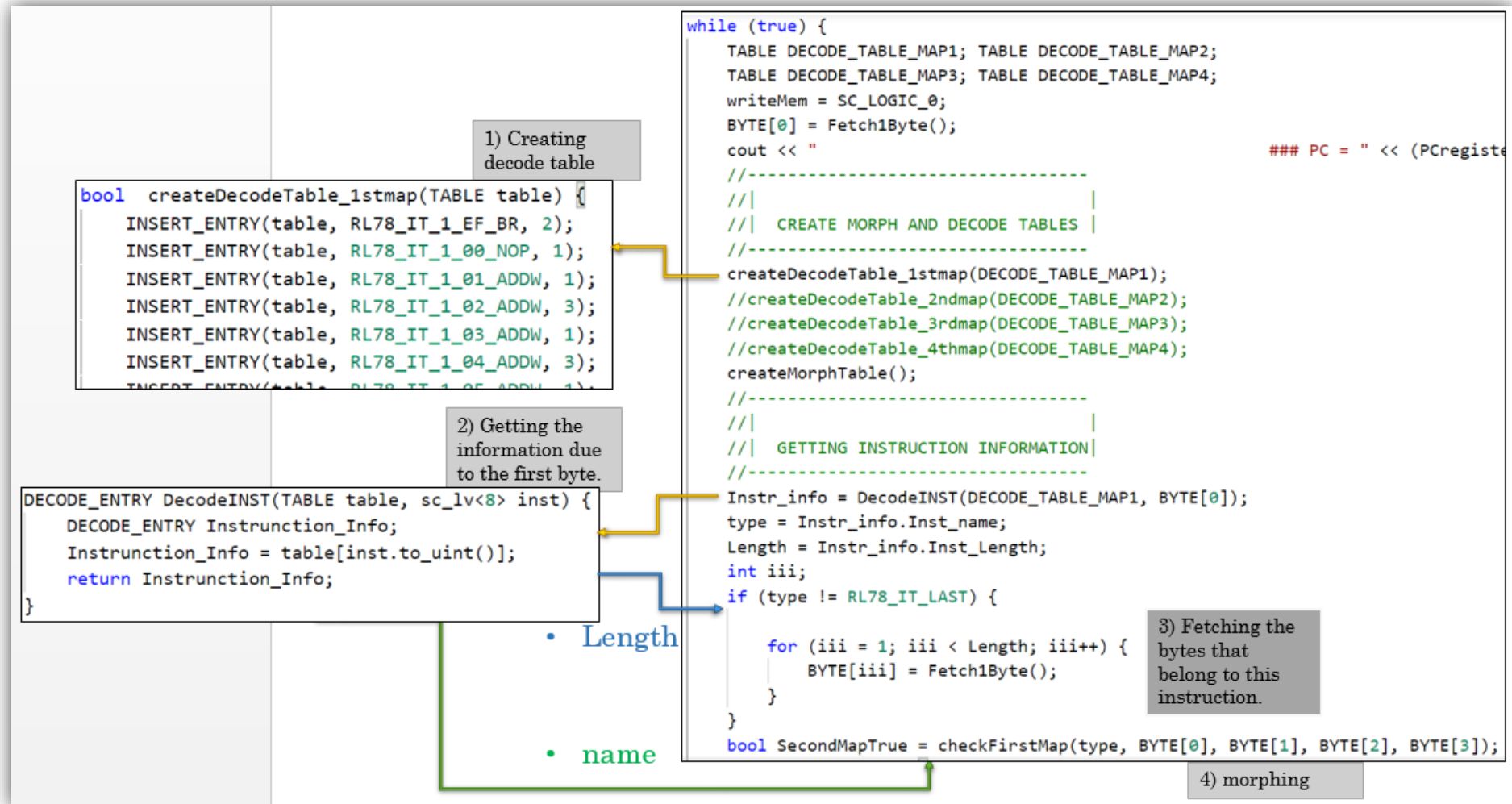


```
SC_MODULE(issRL78) {  
    sc_in <sc_logic> clk;  
    sc_in <sc_logic> memReady;  
  
    sc_inout <sc_lv<8>> dataBus;  
  
    sc_out <sc_logic> readMem, writeMem;  
    sc_out <sc_lv<20>> addrBus;  
  
    struct RL78Struct ST;  
    typedef void (issRL78::*DispathFNC)(sc_lv <8>, sc_lv <8>, sc_lv <8>, sc_lv <8>);  
    DispathFNC MorphFunctions[RL78_IT_LAST];  
  
    SC_HAS_PROCESS(issRL78);  
  
    issRL78(sc_module_name) {  
        ST.adrSpace = int(pow(2, 20));  
        SC_THREAD(abstractSimulation);  
        sensitive << clk.pos();  
    }  
}
```

Flowchart



Decode



Morph

```
while (true) {
    TABLE DECODE_TABLE_MAP1; TABLE DECODE_TABLE_MAP2;
    TABLE DECODE_TABLE_MAP3; TABLE DECODE_TABLE_MAP4;
    writeMem = SC_LOGIC_0;
    BYTE[0] = Fetch1Byte();
    cout << "
    //-----
    //|
    //|  CREATE MORPH AND DECODE T
    //-----
    createDecodeTable_1stmap(DECODE_TABLE_MAP1);
    //createDecodeTable_2ndmap(DECODE_TABLE_MAP2);
    //createDecodeTable_3rdmap(DECODE_TABLE_MAP3);
    //createDecodeTable_4thmap(DECODE_TABLE_MAP4);
    createMorphTable();
    //-----
    //|
    //|  GETTING INSTRUCTION INFORMATION
    //-----
    Instr_info = DecodeINST(DECODE_TABLE_MAP1, BYTE[0]);
    type = Instr_info.Inst_name;
    Length = Instr_info.Inst_Length;
    int iii;
    if (type != RL78_IT_LAST) {
        for (iii = 1; iii < Length; iii++) {
            BYTE[iii] = Fetch1Byte();
        }
    }
    bool SecondMapTrue = checkFirstMap(type, BYTE[0], BYTE[1], BYTE[2], BYTE[3]);
}

void issRL78::createMorphTable() {
    MorphFunctions[RL78_IT_1_00_NOP] = &issRL78::MORPH_1_00_NOP;
    MorphFunctions[RL78_IT_1_01_ADDW] = &issRL78::MORPH_1_01_ADDW;
    MorphFunctions[RL78_IT_1_02_ADDW] = &issRL78::MORPH_1_02_ADDW;
    MorphFunctions[RL78_IT_1_03_ADDW] = &issRL78::MORPH_1_03_ADDW;
    MorphFunctions[RL78_IT_1_04_ADDW] = &issRL78::MORPH_1_04_ADDW;
}

(this->*MorphFunctions[type])(Byte1, Byte2, Byte3, Byte4);
```

Functions used for morphing

- These are some of the functions used for morphing

```
#define ReaddataOffsetSP_16(b, Y)\
sc_lv<16> adr;\
ReadSP(adr);\
sc_lv<8> dl = Read1Byte((adr).to_uint() + b.to_uint() + 0xF0000);\
sc_lv<8> dh = Read1Byte((adr).to_uint() + b.to_uint() + 0xF0000 + 1);\
Y = dh.to_uint()* 256 + dl.to_uint();\
cout<<"data is: "<< Y <<endl;\

#define ReaddataOffset_8(M, b, Y)\
dataA_8b = Read1Byte(Read1Byte(R_MMAP(M)).to_uint() + b.to_uint());\
Y = dataA_8b.to_uint();\
cout<<"data is: "<< Y <<endl;\

#define Immediate2Byte(addr)\
addr = Byte3.to_uint() * 256 + Byte2.to_uint();\
cout<<"addrHL: "<<addr<<endl;\

#define ImmediateData2Byte(addr)\
addr = Byte4.to_uint() * 256 + Byte3.to_uint();\
cout<<"addrHL: "<<addr<<endl;\

#define DO_Addition(op1, op2, result, n)\
result = op1.to_int() + op2.to_int();\
cout << "addition result: " << result << endl;\

#define DO_Multiplication(op1, op2, result)\
result = op1.to_uint() * op2.to_uint();\
cout << "addition result: " << result << endl;\
```

Conclusion

- ✓ RL78 Memory Map and Addressing
- ✓ RL78 Instruction Set
- ✓ ISS Modeling Proposed Methodologies
- ✓ Switch-Case Methodology
- ✓ Morph-Decode Methodology
- ✓ Implementation
- ✓ Work Duration: 7 weeks

Thank you for your attention