ARCHLAB 报告

软件71 唐建宇 2017012221

PART B

指令的实现步骤

IIADDL

Fetch	icode:ifun← M1[PC] rA:rB← M1[PC+1] ValC← M4[PC+2] ValP← PC+6
Decode	ValA← R[rA]
Execute	ValE← ValC+ValA
Memory	
Write back	R[rA]← ValE
PC update	PC← ValP

ILEAVE

Fetch	icode:ifun← M1[PC] rA:rB← M1[PC+1] ValC← M4[PC+2] ValP← PC+1
Decode	ValA← R[%ebp] ValB← R[%ebp]
Execute	ValE← ValB+4
Memory	ValM← M[ValA]
Write back	R[%esp]← ValE R[%ebp]← ValM
PC update	PC← ValP

SEQ处理器HCL文件的修改

按照两个指令按上述方法拆分得到的步骤，在HCL文件中进行对应的修改。涉及到的 HCL 文件修改会在需要修改每行的代码行末注明所进行的修改。

1.取指阶段

```
bool instr_valid = icode in
    { INOP, IHALT, IRRMOVL, IIRMOVL, IRMMOVL, IMRMOVL,
           IOPL, IJXX, ICALL, IRET, IPUSHL, IPOPL, IIADDL, ILEAVE };
    #加入了 IIADDL,ILEAVE

# Does fetched instruction require a regid byte?
bool need_regids =
    icode in { IRRMOVL, IOPL, IPUSHL, IPOPL,
              IIRMOVL, IRMMOVL, IMRMOVL, IIADDL };                    #加入了 IIADDL

# Does fetched instruction require a constant word?
bool need_valC =
    icode in { IIRMOVL, IRMMOVL, IMRMOVL, IJXX, ICALL, IIADDL };      #加入了 IIADDL
```

首先将 IIADDL 和 ILEAVE 两个指令加入 instr_valid 中。

由于 IIADDL 需要寄存器和 ValC 作为参数，因此在 need_regids 和 need_valC 加入IIADDL。

2.译码阶段

```
## What register should be used as the A source?

int srcA = [
    icode in { IRRMOVL, IRMMOVL, IOPL, IPUSHL  } : rA;
    icode in { IPOPL, IRET } : RESP;
        icode in { ILEAVE } : REBP;                          #加入ILEAVE
    1 : RNONE; # Don't need register
];

#What register should be used as the B source?

int srcB = [
    icode in { IOPL, IRMMOVL, IMRMOVL, IIADDL  } : rB;      #加入IIADDL
    icode in { IPUSHL, IPOPL, ICALL, IRET } : RESP;
        icode in { ILEAVE } : REBP;                          #加入ILEAVE
    1 : RNONE;  # Don't need register
];

What register should be used as the E destination?

int dstE = [
    icode in { IRRMOVL } && Cnd : rB;
    icode in { IIRMOVL, IOPL, IIADDL} : rB;                   #加入IIADDL
    icode in { IPUSHL, IPOPL, ICALL, IRET, ILEAVE } : RESP;  #加入ILEAVE
    1 : RNONE;  # Don't write any register
];

What register should be used as the M destination?

int dstM = [
    icode in { IMRMOVL, IPOPL } : rA;
        icode in { ILEAVE } : REBP;                          #加入ILEAVE
    1 : RNONE;  # Don't write any register
];`
```

由于 IIADDL 需要在译码阶段将 rB 寄存器的值读入 ValB 中，因此在 srcB 中的 rB 部分加入 IIADDL；而写回阶段中，需要把 ALU 计算得到的 ValE 值重新写入 rB 寄存器，因此在 dstM 中的 rB 部分加入 IIADDL。

由于 ILEAVE 需要在在译码阶段将 %ebp 寄存器的值读入 ValA 和 ValB 中，因此在 srcA 中的 rA 部分加入 ILEAVE，在 srcB 中的 rB 部分加入 ILEAVE；而写回阶段中，需要把 ALU 计算得到的 ValE 值和访存得到的 ValM 值分别写入 %esp 寄存器和 %ebp 寄存器，因此在 dstE 中的 RESP 部分加入 ILEAVE，在 dstE 中的 REBP 部分加入 ILEAVE。

3.执行阶段

```
## Select input A to ALU
int aluA = [
     icode in { IRRMOVL, IOPL } : valA;

     icode in { IIRMOVL, IRMMOVL, IMRMOVL, IIADDL } : valC;       #加入 IIADDL
```

```
    icode in { ICALL, IPUSHL } : -4;
    icode in { IRET, IPOPL, ILEAVE } : 4;                          #加入 ILEAVE
    # Other instructions don't need ALU
];

## Select input B to ALU
int aluB = [
    icode in { IRMMOVL, IMRMOVL, IOPL, ICALL,
               IPUSHL, IRET, IPOPL, ILEAVE, IIADDL } : valB;      #加入 ILEAVE, IIADDL
    icode in { IRRMOVL, IIRMOVL } : 0;
    # Other instructions don't need ALU
];

## Should the condition codes be updated?
bool set_cc = icode in { IOPL, IIADDL };                           #加入 IIADDL
```

在执行阶段，IIADDL 的操作是将 ValC 和 ValB 相加，因此在 aluA 中的 ValC 部分加入 IIADDL，在 aluB 中的 ValB 部分加入 IIADDL。同时作为加法运算，IIADDL 需要更新条件码，因此在 set_cc 中加入IIADDL 。

ILEAVE 的操作是将 ValB 与 4 相加，因此在 aluA 中的 4 部分加入 ILEAVE，在 aluB 中的 ValB 部分加入 ILEAVE。

4.访存阶段

```
## Set read control signal
bool mem_read = icode in { IMRMOVL, IPOPL, IRET, ILEAVE };      #加入ILEAVE

## Select memory address
int mem_addr = [
    icode in { IRMMOVL, IPUSHL, ICALL, IMRMOVL } : valE;
    icode in { IPOPL, IRET, ILEAVE } : valA;                    #加入ILEAVE
    # Other instructions don't need address
];
```

在访存阶段，IIADDL 无需操作，只有 ILEAVE 需要读取内存上 ValA 地址上的数据，即 M1[ValA]。因此在 mem_read 中加入 ILEAVE，在 mem_addr 中加入地址为 ValA 的部分加入 ILEAVE。

5.PC更新阶段

因为 ILEAVE 和 IIADDL 两个指令本身都不涉及跳转，下一条执行的地址均为默认值 ValP ，因此这一部分代码不需要修改。

PIPELINE处理器HCL文件的修改

流水线处理器的修改部分与顺序执行处理器类似，同样按照拆分的步骤进行修改。

1.取指阶段

```
# Is instruction valid?
bool instr_valid = f_icode in
    { INOP, IHALT, IRRMOVL, IIRMOVL, IRMMOVL, IMRMOVL,
      IOPL, IJXX, ICALL, IRET, IPUSHL, IPOPL, IIADDL, ILEAVE };

# Does fetched instruction require a regid byte?
bool need_regids =
    f_icode in { IRRMOVL, IOPL, IPUSHL, IPOPL,
                 IIRMOVL, IRMMOVL, IMRMOVL, IIADDL };

# Does fetched instruction require a constant word?
bool need_valC =
    f_icode in { IIRMOVL, IRMMOVL, IMRMOVL, IJXX, ICALL, IIADDL };
```

首先将 IIADDL 和 ILEAVE 两个指令加入 instr_valid 中。

由于 IIADDL 需要寄存器和 ValC 作为参数，因此在 need_regids 和 need_valC 加入IIADDL。

在预测PC的逻辑中，因为两个指令均不涉及跳转，因此无需改动采用默认的 f_valP 即可。

2.译码阶段

```
## What register should be used as the A source?
int d_srcA = [
    D_icode in { IRRMOVL, IRMMOVL, IOPL, IPUSHL  } : D_rA;
    D_icode in { IPOPL, IRET } : RESP;
        D_icode in { ILEAVE } : REBP;
    1 : RNONE; # Don't need register
];

## What register should be used as the B source?
int d_srcB = [
    D_icode in { IOPL, IRMMOVL, IMRMOVL, IIADDL  } : D_rB;
    D_icode in { IPUSHL, IPOPL, ICALL, IRET } : RESP;
        D_icode in { ILEAVE } : REBP;
    1 : RNONE;  # Don't need register
];

## What register should be used as the E destination?
int d_dstE = [
    D_icode in { IRRMOVL, IIRMOVL, IOPL, IIADDL } : D_rB;
    D_icode in { IPUSHL, IPOPL, ICALL, IRET, ILEAVE } : RESP;
    1 : RNONE;  # Don't write any register
];

## What register should be used as the M destination?
int d_dstM = [
    D_icode in { IMRMOVL, IPOPL } : D_rA;
        D_icode in { ILEAVE } : REBP;
    1 : RNONE;  # Don't write any register

];
```

由于 IIADDL 需要在译码阶段将 rB 寄存器的值读入 ValB 中，因此在 srcB 中的 rB 部分加入 IIADDL；而写回阶段中，需要把 ALU 计算得到的 ValE 值重新写入 rB 寄存器，因此在 dstM 中的 rB 部分加入 IIADDL。

由于 ILEAVE 需要在在译码阶段将 %ebp 寄存器的值读入 ValA 和 ValB 中，因此在 srcA 中的 rA 部分加入 ILEAVE，在 srcB 中的 rB 部分加入 ILEAVE；而写回阶段中，需要把 ALU 计算得到的 ValE 值和访存得到的 ValM 值分别写入 %esp 寄存器和 %ebp 寄存器，因此在 dstE 中的 RESP 部分加入 ILEAVE，在 dstE 中的 REBP 部分加入 ILEAVE。

3.执行阶段

```
## Select input A to ALU
int aluA = [
    E_icode in { IRRMOVL, IOPL } : E_valA;
    E_icode in { IIRMOVL, IRMMOVL, IMRMOVL, IIADDL } : E_valC;
    E_icode in { ICALL, IPUSHL } : -4;
    E_icode in { IRET, IPOPL, ILEAVE } : 4;
    # Other instructions don't need ALU
];

## Select input B to ALU
int aluB = [
    E_icode in { IRMMOVL, IMRMOVL, IOPL, ICALL,
                 IPUSHL, IRET, IPOPL, IIADDL, ILEAVE } : E_valB;
    E_icode in { IRRMOVL, IIRMOVL } : 0;
    # Other instructions don't need ALU
];

## Should the condition codes be updated?
bool set_cc = (E_icode == IIADDL) || (E_icode == IOPL &&
    # State changes only during normal operation
    !m_stat in { SADR, SINS, SHLT } && !W_stat in { SADR, SINS, SHLT });
```

在执行阶段，IIADDL 的操作是将 ValC 和 ValB 相加，因此在 aluA 中的 ValC 部分加入 IIADDL，在 aluB 中的 ValB 部分加入 IIADDL。同时作为加法运算，IIADDL 需要更新条件码，因此在 set_cc 中加入IIADDL ，由于原先只有 E_icode 为 IOPL 且满足一些条件时才需要更新条件码，因此在这里将 `E_icode ==IIADDL` 与 `E_icode == IOPL && !m_stat in { SADR, SINS, SHLT } && !W_stat in { SADR, SINS, SHLT }` 用 **或** 的逻辑连接。

ILEAVE 的操作是将 ValB 与 4 相加，因此在 aluA 中的 4 部分加入 ILEAVE，在 aluB 中的 ValB 部分加入 ILEAVE。

4.访存阶段

```
## Select memory address
int mem_addr = [
    M_icode in { IRMMOVL, IPUSHL, ICALL, IMRMOVL } : M_valE;
    M_icode in { IPOPL, IRET, ILEAVE } : M_valA;
    # Other instructions don't need address
];

## Set read control signal
bool mem_read = M_icode in { IMRMOVL, IPOPL, IRET, ILEAVE };
```

在访存阶段，IIADDL 无需操作，只有 ILEAVE 需要读取内存上 ValA 地址上的数据，即 M1[ValA]。因此在 mem_read 中加入 ILEAVE，在 mem_addr 中加入地址为 ValA 的部分加入 ILEAVE。

5.PC更新阶段

因为 ILEAVE 和 IIADDL 两个指令本身都不涉及跳转，下一条执行的地址均为默认值 ValP ，因此这一部分代码不需要修改。