第4章 处理器体系结构

Y86-64指令集

- ISA(Instruction-Set Architecture): 一个处理器支持的指令和指令的字节级编码
- 程序员可见的状态
  - 15 registers: %rax, %rcx, %rdx, %rbx, %rsp, %rbp, %rsi, %rdi, %r8-%r14
  - CC(Condition codes): ZF(是否为0), SF(负数), OF(无符号溢出)
    - OPq指令会设置CC
  - PC(Program counter)
  - memory
  - Stat: 表明程序正常运行，或是出现某种异常

value	name	含义
1	AOK	正常操作
2	HLT	执行halt指令
3	ADR	遇到非法地址
4	INS	遇到非法指令

    出现异常时，处理器会调用exception handler
- Y86-64指令及编码
  - 详见Computer Systems-A programmer's perspective P357-P359
- CISC v.s. RISC

CISC	RISC
指令种类多	指令种类少，但编译出来指令多
实现细节不可见	实现细节可见
--	流水化作业
编码长度可变	编码固定长度
有条件码	无条件码
栈用来存取过程参数和返回地址	寄存器被用来存取过程参数和返回地址
- directive伪指令
  - .data: .byte .word .long .quad
  - .pos
  - .align
- convension
  - pushq压入%rsp的原始值
  - popq取出%rsp栈中的值

HCL(Hardware Control Language)

- HCL表达硬件设计的控制部分，已经开发有将HCL直接翻译成Verilog的工具，将这个代码与Verilog结合起来就能产生HDL，再据此就可以合成微处理器
- Logic Gates: AND&& OR|| NOT!
- Combinational Circuits and HCL Boolean Expressions `bool eq = (a && b) || (!a && !b)` `bool mux = (s && a) || (!s && b)`
- Word-Level Combinational Circuits and HCL Integer Expressions `bool Eq = (A == B)`

```
word Mux = [
    s: A;
    1: B;
];
word Min3 = [
    A <= B && A <= C : A;
    B <= A && B <= C : B;
    1                : C;
];
```

- Set Membership `bool s = code in { 1, 3 }`
- Memory and Clocking
  - register file
  - random access memory

Sequential Y86-64

- 指令六个阶段
  - fetch: icode:ifun [rA:rB] [valC] valP
  - decode: [valA] [valB] (%rsp)
  - execute: (只有OPq设CC)
    - valE <- valB/0 + valA/valC/8/-8
    - Set CC / Cnd <- Cond(CC, ifun)
  - memory: <valM, valA, valE, valP>
  - write back: <valE, valM>
  - PC update: <valP, valC, valM>
- SEQ硬件结构
  - 详见Computer Systems-A programmer's perspective P397-P400
- SEQ时序
  - 组合逻辑: Instruction memory可以视作组合逻辑
  - 存储器设备
    - clocked register(PC, CC)
    - RAM(register file, instruction memory, data memory)
  - 原则： No reading back
- SEQ Stage 实现
  - Fetch Stage `bool imem_error = ...` `bool instr_valid = icode in {...}` `bool need_regids = icode in { IRRMOVQ, IOPQ, IPUSHQ, IPOPQ, IIRMOVQ, IRMMOVQ, IMRMOVQ };` `bool need_valC = icode in { IIRMOVQ, IRMMOVQ, IMRMOVQ, IJXX, ICALL}`
  - Decode and Write-Back Stages

```
word srcA = [
    icode in { IRRMOVQ, IRMMOVQ, IOPQ, IPUSHQ } : rA;
    icode in { IPOPQ, IRET } : RRSP;
    1 : RNONE;
];
```

```
word srcB = [
    icode in { IOPQ, IRMMOVQ, IMRMOVQ } : rB;
    icode in { IPUSHQ, IPOPQ, ICALL, IRET } : RRSP;
    1 : RNONE;
];
```

```
word dstE = [
    icode in { IRRMOVQ } : rB;
    icode in { IIRMOVQ, IOPQ } : rB;
    icode in { IPUSHQ, IPOPQ, ICALL, IRET } : RRSP;
    1 : RNONE;
];
```

```
word dstM = [
    icode in { IMRMOVQ, IPOPQ } : rA;
    1 : RNONE;
]
```

- Execute Stage

```
word aluA = [
    icode in { IRRMOVQ, IOPQ } : valA;
    icode in { IIRMOVQ, IRMMOVQ, IMRMOVQ } : valC;
    icode in { ICALL, IPUSHQ } : -8;
    icode in { IRET, IPOPQ } : 8;
];
```

```
word aluB = [
    icode in { IRMMOVQ, IMRMOVQ, IOPQ, ICALL, IPUSHQ, IRET, IPOPQ } :
valB;
    icode in { IRRMOVQ, IIRMOVQ } : 0;
];
```

```

```

```
word alufun = [
    icode == IOPQ : ifun;
    1 : ALUADD;
];
```

```
bool set_cc = icode in { IOPQ };
```

- Memory Stage

```
word mem_addr = [
    icode in { IRMMOVQ, IPUSHQ, ICALL, IMRMOVQ } : valE;
    icode in { IPOPQ, IRET } : valA;
];
```

```
word mem_data = [
    icode in { IRMMOVQ, IPUSHQ } : valA;
    icode == ICALL : valP;
]
```

```
bool mem_read = icode in { IMRMOVQ, IPOPQ, IRET }; bool mem_write = icode in { IRMMOVQ, IPUSHQ, ICALL };
```

```
word Stst = [
    imem_error || dmem_error : SADR;
    !instr_valid : SINS;
    icode == IHALT : SHLT;
    1 : SAOK;
];
```

- PC Update Stage

```
word new_pc = [
    icode == ICALL : valC;
    icode == IJXX && Cnd : valC;
    icode == IRET : valM;
    1 : valP;
];
```

Pipeline

- Throughput: 每秒执行指令数，单位GIPS(每秒千兆条指令)
- latency: 从头到尾执行一条指令所需要的时间
- Limitations:

    1. Nonuniform Partitioning 不一致的划分
    2. Diminishing Returns of Deep Pipelining 流水线过深，收益反而下降
- Feedback:
    - data dependency
    - control dependency

Pipelined Y86-64 Implementations

- SEQ+ 移动PC Update，合并时钟
- PIPE- 增加5个pipeline registers
- FDEMW前缀是流水线寄存器，fdemw前缀是流水线阶段
- Next PC Prediction: 从valP和valC选择出predPC, jXX预测taken，ret不预测
- Hazard
    - data hazard
    - control hazard
    - load/use hazard
- Avoiding
    - Stalling
    - Forwarding
- Exception Handling
    - 多个异常发生时应该报告流水线中最深的异常
    - 分支预测错误导致的异常应该能够取消
    - 出现异常后的指令不应该修改系统的部分状态
- PIPE
    - PC Selection and Fetch Stage

```
word f_pc = [
    M_icode == IJXX && !M_Cnd : M_valA;
    W_icode == IRET : W_valM;
    1 : F_predPC;
];
```

```
word f_predPC = [
    f_icode in { IJXX, ICALL } : f_valC;
    1 : f_valP;
]
```

```
word f_stat = [
    imem_error : SADR;
    !instr_valid : SINS;
    f_icode == IHALT : SHLT;
    1 : SAOK;
]
```

- Decode and Write-Back Stage

```
word d_dstE = [
    D_icode in { IRRMOVQ, IIRMOVQ, IOPQ } : D_rB;
    D_icode in { IPUSHQ, IPOPQ, ICALL, IRET } : RRSP;
    1 : RNONE;
]
```

```
word d_valA = [
    D_icode in { ICALL, IJXX } : D_valP;
    d_srcA == e_dstE : e_valE;
    d_srcA == M_dstM : m_valM;
    d_srcA == M_dstE : M_valE;
    d_srcA == W_dstM : W_valM;
    d_srcA == W_dstE : W_valE;
    1 : d_rvalA;
]
```

```
word d_valB = [
    d_srcB == e_dstE : e_valE;
    d_srcB == M_dstM : m_valM;
    d_srcB == M_dstE : M_valE;
    d_srcB == W_dstM : W_valM;
    d_srcB == W_dstE : W_valE;
    1 : d_rvalB;
];
```

```
word Stat = [
    W_stat == SBUB : SAOK;
    1 : W_stat;
]
```

- Execute Stage
- Memory Stage

```
word m_stat = [
    dmem_error : SADR;
    1 : M_stat;
]
```

- Pipeline Control Logic
    - Load/use harzard, 从内存读取和使用该值的指令之间必须暂停 1 cycle
        - E_icode in {IMRMOVQ, IPOPQ} && E_dstM in {d_srcA, d_srcB}
    - Processing ret，必须stall直到ret到达write-back阶段
        - IRET in {D_icode, E_icode, M_icode}
    - Mispredicted branche，必须取消这些指令，从新的指令开始
        - E_icode == IJXX && !e_Cnd
    - Exception，禁止后面的指令更新状态，并在异常指令到达write-back时停止执行
        - m_stat in {SADR, SINS, SHLT} || W_stat in {SADR, SINS, SHLT}

Condition	**F**	**D**	**E**	**M**	**W**
Precessing ret	stall	bubble	normal	normal	normal
Load/use hazard	stall	stall	bubble	normal	normal
Mispredicted branch	normal	bubble	bubble	normal	normal

详见P460-462 `bool F_stall = E_icode in { IMRMOVQ, IPOPQ } && E_dstM in { d_srcA, d_srcB } || IRET in { D_icode, E_icode, M_icode };` 其他代码详见P493-494

- 性能分析
    - CPI(Cycles Per Instruction) = 1 + lp + mp + rp