ICS Homework 5

March 24, 2021

1 Pipeline

Suppose we add a new instruction *rjmp rB* to Y86 instruction sets. It will jmp to the address stored in *rB*.
1. Fill in the function of each stage for *rjmp rB* instructionin Y86 sequential implementation. (NOTE: use *valB* to update PC)

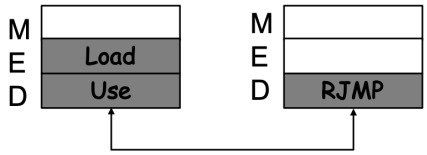
Stage	*rjmp rB*
F	icode:ifun <- M1[PC] rA:rB <- M1[PC+1] valP <- PC + 2
D	valB <- R[rB]
E	–
M	–
W	–
P	PC <- valB

2. As shown in the new PIPE logic figure, we add a forwarding logic from *E_valB* to *f_pc* to support *rjmp* instruction, since the target address require read from register file. Please describe all possible hazards due to new instruction *rjmp*. You need provide detail explanation and list detection conditions like Figure 4.64 and control action like Figure 4.66. (Do not consider the hazard combinations here.)

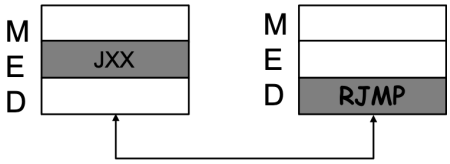
Condition	Trigger
Target-addr Hazard	IRJMP in {D_icode}

F	D	E	M	W
Stall	Bubble	Normal	Normal	Normal

3. Please list all hazard combinations (arise simultaneously) including *rjmp* instruction. You need draw pipeline states figures like Figure 4.67 and list pipeline control action like tables after Figure 4.67 for each combination.



Condition	F	D	E	M	W
Load	Stall	Stall	Bubble	Normal	Normal
RJMP	Stall	Bubble	Normal	Normal	Normal
Combination	Stall	B+S	Bubble	Normal	Normal
Desired	Stall	Stall	Bubble	Normal	Normal



Condition	F	D	E	M	W
JXX	Normal	Bubble	Bubble	Normal	Normal
RJMP	Stall	Bubble	Normal	Normal	Normal
Desired	S or B or N	Bubble	Bubble	Normal	Normal

4.The original PIPE implementation of Y86 should be modified to support the *rjmp* instruction. Please describe the modification and provide HCL of *f_pc* and *D_bubble* logic. (NOTE: Only need to write the code about *rjmp* instruction)

```
word f_pc = [
    E_icode == IRJMP : E_valB
];


bool D_bubble = [
    D_icode == IRJMP && !{E_icode in { IMRMOVQ, IPOPQ } &&
E_dstM in { d_srcA, d_srcB}}
];
```

2 Machine Independent Optimization

Suppose we have some codes as below.

```
1
2 typedef struct {
3     int vals[3];
```

```
} block_t;

typedef struct {
    int length;
    block_t *blocks;
} blocklist;

int get_length(blocklist *bl) {
    return bl->length;
}

block_t* get_blocks(blocklist *bl) {
    return bl->blocks;
}

void SUM(blocklist *bl, long *dest) {
    for (int i = 0; i < get_length(bl); i++) {
        int size = 1;
        for (int j = 0; j < 3; j++)
            size = size * get_blocks(bl)[i].vals[j];
            *dest = *dest + size;
        }
}
```

Try to optimize the function SUM with a combination of optimizations you have learned in the ICS class. Comment briefly on your optimizations.

```
void SUM(blocklist *bl, long *dest) {
    // reduce loop overhead
    int length = get_length(bl);

    //reduce function call
    block_t* blocks = get_blocks(bl);

    int tmp = 0;
    for (int i = 0; i < get_length(bl); i++) {

        // reduce repeated calculation
        int* vals = blocks[i].vals;

        //loop unrolling
        tmp = tmp + vals[0]*vals[1]*vals[2];
    }

    //reduce memory access
    *dest = tmp;
```

```
}
```

3 Signal

```
int counter = 2;

void handler1(int sig) {
    counter = counter + 1;
    printf("%d\n", counter);
    exit(0);
}

int main() {
    signal(SIGINT, handler1);
    printf("%d\n", counter);
    if ((pid = fork()) == 0) {
        while(1) {};
    }
    kill(pid, SIGINT);

    counter = counter -1;
    printf("%d\n", counter);
    waitpid(-1, NULL, 0);
    counter = counter + 1;
    printf("%d\n", counter);
    exit(0);
}
```

1. Please rewrite the *handler* according to the guidelines in section 8.5.5 (HINT: you can use *Sio_puts* as thread safe *printf* if needed).

```
volatile sig_atomic_t counter = 2;

void handler1(int sig) {
    int olderrorno = errorno;
    sigset_t mask, prev_mask;
    Sigfillset(&mask);
    Sigprocmask(SIG_BLOCK, &mask, &prev_mask);
    counter = counter + 1;
    Sio_putf("%d\n", counter);
    Sigprocmask(SIG_BLOCK, &prev_mask, NULL);
    errorno = olderrorno;
    _exit(0);
```

```
13 }
14
15 int main {
16     .........
```

2. Please write down all the possible outputs of the original programs.
   2\n3\n1\n2\n or 2\n1\n3\n2\n