Pipelined Beta: 2

Bookmark this page

For all Beta related questions, you should make use of the Beta documentation, the Beta Instruction Summary, the Unpipelined Beta Diagram and the Pipelined Beta Diagram.

Pipelined Beta: 2

0.8823529411764706/1 point (ungraded)

Recall the code for gcd that we saw in lecture, and the assembly code for the while loop:

C code:

```
  int gcd(int x, int y) {
    while (x != y) {
      if (x > y) {
        x = x - y;
      }
      else {
        y = y - x;
      }
    }
    return x;
}
```

Corresponding Beta assembly for while loop:

```
      // x in R0, y in R1
      CMPEQ(R0, R1, R2)     // R2 <-- (x == y)
      BT(R2, end)
loop: CMPLT(R1, R0, R2)     // R2 <-- (x > y)
      BF(R2, else)
      SUB(R0, R1, R0)       // x <-- x - y
      BR(cond)
else: SUB(R1, R0, R1)       // y <-- y - x
cond: CMPEQ(R1, R0, R2)     // R2 <-- (x == y)
      BF(R2, loop)
end:  ...
```

Assume a **5-stage pipelined Beta** as presented in lecture, with **full bypass paths**, and which **predicts branches by assuming they are not taken** to resolve control (i.e., the instruction following the branch is fetched in the IF stage on the cycle after the branch is in the IF stage).

First, find the number of cycles per iteration in steady state (do not worry about the first or last iterations). Note that the BF(R2, else) branch is not taken if x > y and taken if x < y, so you should consider these two cases separately.

1. Fill in the following table:

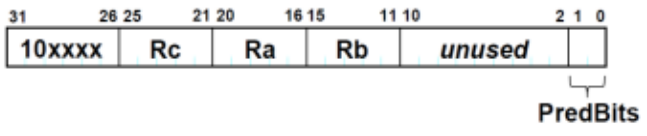
	Iterations where x > y	Iterations where x < y
Instructions per iteration	6 ✔ Answer: 6	5 ✔ Answer: 5
+ Cycles lost to data hazards	0 ✔ Answer: 0	0 ✔ Answer: 0
+ Cycles lost to annulments	2 ✔ Answer: 2	2 ✔ Answer: 2
= Total cycles per iteration	8 ✔ Answer: 8	7 ✔ Answer: 7

Explanation
The loop instructions executed if x > y are CMPLT, BF, SUB, BR, CMPEQ, BF or 6 instructions. The loop instructions executed if x < y are CMPLT, BF, SUB, CMPEQ, BF or 5 instructions. There are no data hazards in either case because full bypass paths are available and none of the hazards are related to a LD instruction. The number of annullments required if x > y is 2 to annul the SUB after the BR(cond) and to annul the instruction fetched after the BF(R2, loop). The number of annullments required if x < y is also 2 to annul the SUB after the BF(R2, else) and to annul the instruction fetched after the BF(R2, loop).

To make this code faster, we modify the Beta ISA and pipeline to implement a technique called **predication** to reduce the number of branches.

First, all the compare instructions (CMPEQ, CMPLT, CMPLE, and their C variants) write their result into a special 1-bit register, called the predicate register, in addition to their normal destination register.

Second, we change the format of ALU instructions with two register source operands to use their lower two bits, which were previously unused:



- If PredBits == 10, the instruction only executes if the predicate register is false (0)
- If PredBits == 11, the instruction only executes if the predicate register is true (1)
- If PredBits == 0X, the instruction always executes and writes its result, as before

We say that instructions that depend on the predicate register are predicated. We denote predicated instructions in assembly as follows:

- If PredBits == 10, OP(Ra, Rb, Rc) [predFalse]
- If PredBits == 11, OP(Ra, Rb, Rc) [predTrue]
- If PredBits == 0X, OP(Ra, Rb, Rc), as before

For example, consider the following instruction sequence:

```
        CMPLT(R1,R2,R3)
        MUL(R3,R4,R5)
        ADD(R4,R5,R6)[predTrue]
        SUB(R5,R6,R7)
```

If the CMPLT instruction evaluates to true (i.e., writes 1 to R3), this sequence is equivalent to:

```
    CMPLT(R1,R2,R3)
    MUL(R3,R4,R5)
    ADD(R4,R5,R6)
    SUB(R5,R6,R7)
```

If the CMPLT instruction evaluates to false (i.e., writes 0 to R3), this sequence is equivalent to:

```
    CMPLT(R1,R2,R3)
    MUL(R3,R4,R5)
    SUB(R5,R6,R7)
```

The following code can be modified to use predication as follows:

Original code:

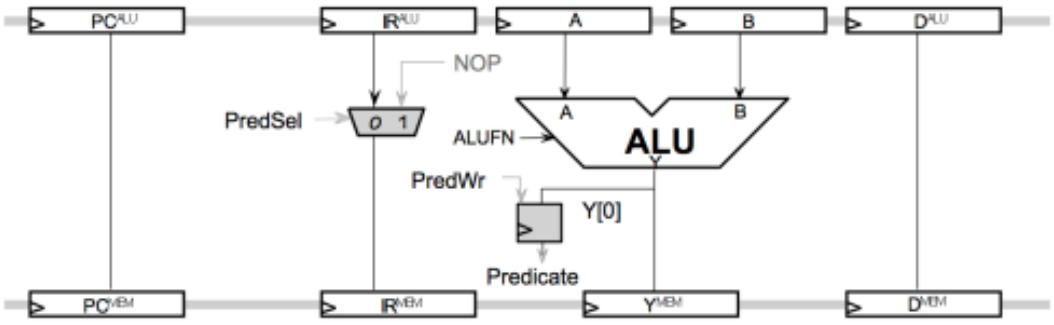
```
      // x in R0, y in R1
      CMPEQ(R0, R1, R2)     // R2 <-- (x == y)
      BT(R2, end)
loop: CMPLT(R1, R0, R2)     // R2 <-- (x > y)
      BF(R2, else)
      SUB(R0, R1, R0)       // x <-- x - y
      BR(cond)
```

```
else: SUB(R1, R0, R1)       // y <-- y - x
cond: CMPEQ(R1, R0, R2)     // R2 <-- (x == y)
      BF(R2, loop)
end:  …
```

Code with predication:

```
      // x in R0, y in R1
      CMPEQ(R0, R1, R2)            // R2 <-- (x == y)
      BT(R2, end)
loop: CMPLT(R1, R0, R2)            // R2 <-- (x > y)
      SUB(R0, R1, R0)[predTrue]
      SUB(R1, R0, R1)[predFalse]
      CMPEQ(R1, R0, R2)
      BF(R2, loop)
end:  ...
```

We implement predication in the pipelined Beta with minor changes to the ALU stage:



Comparison instructions write the 1-bit predicate register (the PredWr control signal ensures that only comparison instructions update the register). The PredSel mux annuls ALU instructions if they are predicated and should not execute according to the value of the predicate register.

2. Select the correct Boolean expression for the PredSel control signal.

- ○ PredSel = (IR_ALU[31:30] == 0b10) AND predBit[0]==1 AND predBit[1]!=Predicate
- ○ PredSel = (IR_ALU[31:30] == 0b10) AND predBit[1]==0 AND predBit[1]!=Predicate
- ● PredSel = (IR_ALU[31:30] == 0b10) AND predBit[1]==1 AND predBit[0]!=Predicate
- ○ PredSel = (IR_ALU[31:30] == 0b10) AND predBit[0]==0 AND predBit[1]==Predicate

✔

Explanation
PredSel is used to determine when to annul an instruction that is currently in the ALU stage. This needs to occur whenever the predicate value is not equal to predBit[0], and predBit[1] = 1. Otherwise, the instruction should proceed as usual.

3. How fast is this modified code? Fill in the following table:

	Iterations where x > y		Iterations where x < y	
Instructions per iteration	4 ✔	Answer: 4	4 ✔	Answer: 4
+ Cycles lost to data hazards	0 ✔	Answer: 0	0 ✔	Answer: 0
+ Cycles lost to annulments	2	Answer: 2	2	Answer: 2
= Total cycles per iteration	6 ✔	Answer: 6	6 ✔	Answer: 6

Explanation
The loop instructions executed in both cases is 4 because only one of the predicated instructions is

The loop instructions executed in both cases is 4 because only one of the predicated instructions is executed in each case. As before there are no data hazards in either case because full bypass paths are available and none of the hazards are related to a LD instruction. The number of annullments required in both cases is 2. One to annul the predicated instruction that is not executed, and another to annul the instruction fetched after the BF(R2, loop).

Submit

Discussion

Hide Discussion

< Previous Next >