Pipelining

CS 217

Instruction Processing Steps

• Instruction fetch: Fetch and decode instruction, retrieve operands from registers
• Execute: Execute arithmetic instruction, compute branch target address, compute load/store memory address
• Memory access: Access memory for load or store, Fetch instruction at target of branch
• Store results: Write instruction results to registers
**Pipelining**

<table>
<thead>
<tr>
<th>Fetch</th>
<th>Execute</th>
<th>Memory</th>
<th>Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch</td>
<td>Execute</td>
<td>Memory</td>
<td>Store</td>
</tr>
<tr>
<td>Fetch</td>
<td>Execute</td>
<td>Memory</td>
<td>Store</td>
</tr>
</tbody>
</table>

```
12  add  %i1, %i1, %o1  PC  nPC
16  add  %i1, %o1, %o1
20  sub  %o1, 3, %o1
24  add  %o1, %i2, %o1
```

**Pipelined Load Instructions**

- Problem: load followed by use

```
ld  [%o0], %o1  load delay slot
add  %o1, %o2, %o2
```

Load delay slots are inserted automatically.
Pipelined Branch Instructions

- Problem: instruction after branch

```
    cmp %0, %1
    ble L1

    branch delay slot

    mov %0, %1
```

```
L1: add %0, %0, %0
```

Updating the Program Counter

- Fetch instruction at address stored in nPC
  - Most instructions: nPC = PC + 4
  - Branch instructions: nPC is computed in execute stage

- Execute instruction at address stored in PC
  - After execute: PC = nPC

```
12   cmp a, b          PC 12 16
16   ble L1           16  20
20   nop               20  36
24   mov a, c          24   40
28   ba L2            28   44
32   nop
36   L1: mov b, c     36  40
40   L2: ...          40  44
```
Delay Slots

- One option: use \texttt{nop} in all delay slots

\begin{verbatim}
for (i=0; i<n; i++)
  ...
  #define i %10
  #define n %11
  clri
  L1: cmp i,n
      bge L2; nop
      ...
      inc i
      bai; nop
\end{verbatim}

Delay Slots

- Optimizing compilers try to avoid delay slots

\begin{verbatim}
for (i=0; i<n; i++)
  ...
  #define i %10
  #define n %11
  clri
  L1: cmp i,n
      bge L2; nop
      ...
      inc i
      bai; nop
  L1: ...
  inc i
  L2: cmp i,n
      bl L1; nop
\end{verbatim}
Delay Slots

- Optimizing compilers try to fill delay slots

```c
if (a > b) c = a; else c = b;

    cmp a, b         cmp a, b
    ble L1;          ble L1
    nop              mov b, c
    mov a, c         mov a, c
    b a L2;          L1: ...
    nop
L1: mov b, c
L2: ...
```

Pipelined Branch Instructions

- Problem: instruction after branch

```c
    cmp %o0, %o1
    ble L1
    mov %o0, %o1

    L1: add %o0, %o0, %o0
```

branch delay slot
Pipelinend Branch Instructions

• Problem: instruction after branch

Programmer should try to insert independent instructions in branch delay slots

Annul Bit

• Controls the execution of the delay-slot instruction

the \texttt{\texttt{a}}, causes the \texttt{mov} instruction to be executed if the branch is taken, and not executed if the branch is not taken

• Exception

\texttt{ba, a} \texttt{L} does \textbf{not} execute the delay-slot instruction
Annul Bit (cont)

- Optimized for (i=0; i<n; i++) 1; 2; ...; n

```
clr i                  clr i
ba L2                 ba, a L2
L1: 1                  L1: 2
2
...                   ...
n
inc i                 inc i
L2: cmp i, n          L2: cmp i, n
bl L1                 bl, a L1
nop
```

While-Loop Example

```
while (...) {
    stmt_t
    : 
    stmt_n
}
```

```
    test: cmp ...
    bx done
    nop
    stmt_t
    : 
    stmt_n
    ba test
    nop
    done: ...  
```

3 instr

2 instr
While-Loop (cont)

• Move test to end of loop

  test:  cmp ...  
  bx done
  nop
  loop: stmt₁
        :
        stmtₙ
  cmp ...  
  bnx loop
  nop
  done: ...

• Eliminate first test

  ba test
  nop
  loop: stmt₁
        :
        stmtₙ
  test:  cmp ...  
  bnx loop
  nop
  ...

While-Loop (cont)

• Eliminate the **nop** in the loop

  ba test
  nop
  loop: stmt₂
        :
        stmtₙ
  test:  cmp ...  
  bnx, a loop
  stmt₁
  ...

  now 2 overhead instructions per loop
If-Then-Else Example

if (...) {
    t-stmt₁
    :
    t-stmtₙ
} else {
    e-stmt₁
    :
    e-stmtₘ
}

How optimize?

if (...) {
    t-stmt₁
    :
    t-stmtₙ
} else {
    e-stmt₁
    :
    e-stmtₘ
}

How optimize?
If-Then-Else Example

if (...) {
    t-stmt₁
    :
    t-stmtₙ
}
else {
    e-stmt₁
    :
    e-stmtₘ
}

How optimize?

cmp ...
bn, a else
e-stmt₁
t-stmt₁
    :   
ba next
t-stmtₙ
else: e-stmt₂
    :   
e-stmtₘ
next: ...