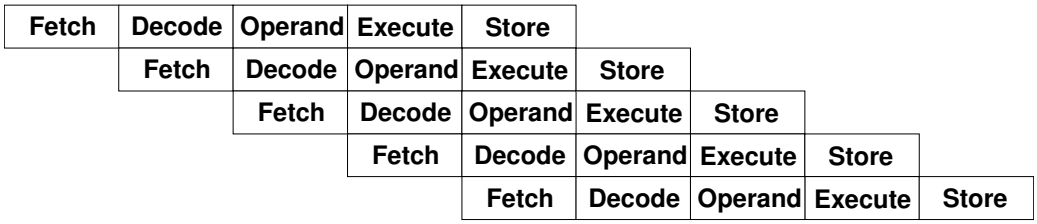
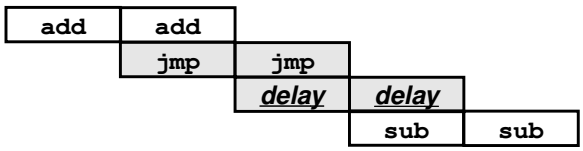
# Instruction Pipelining

- Instruction Pipeline




- ***PC is incremented by*** 4 at the Fetch stage to fetch the next instruction
- A delay slot caused by a jmp instruction, why?

| *PC* | *PC'* | *instruction* |
|---|---|---|
| 8 | 12 | `add` |
| 12 | 16 | `jmp 40` |
| 16 | 40 | ***delay slot*** |
| ... | | ... |
| 40 | 44 | `sub` |








# Delay-Slot Instructions

- If this is a "feature," it certainly contradicts "normal" expectations

  If you think this is confusing, use a **`nop`** in all delay slots

- Optimizers may be able take advantage of the delay slot

  unoptimized **`if (a > b) c = a; else c = b;`** takes 7 cycles

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

  optimized takes 4 cycles

```
    cmp a,b
    ble L1
    mov b,c
    mov a,c
L1:
```

# Annul Bit

- ***Annul bit*** controls the execution of the delay-slot instruction

```
bg,a    L1
mov     a,c
```

the "`,a`" causes the `mov` instruction to be executed if the branch is taken,

and not executed if the branch is not taken

- Exception

`ba,a L` does ***not*** execute its delay-slot instruction

- **What is the advantage of this counterintuitive convention?**





# Annul Bit, cont'd

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

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

n + 4 instructions/iteration

- **better code uses delay slots**

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

n + 3 instructions/iteration

- **Convention of programming**

don't use annul unless absolutely necessary

place nop after control-transfer instructions

- **What happens when the delay-slot instruction is a control-transfer instruction?**