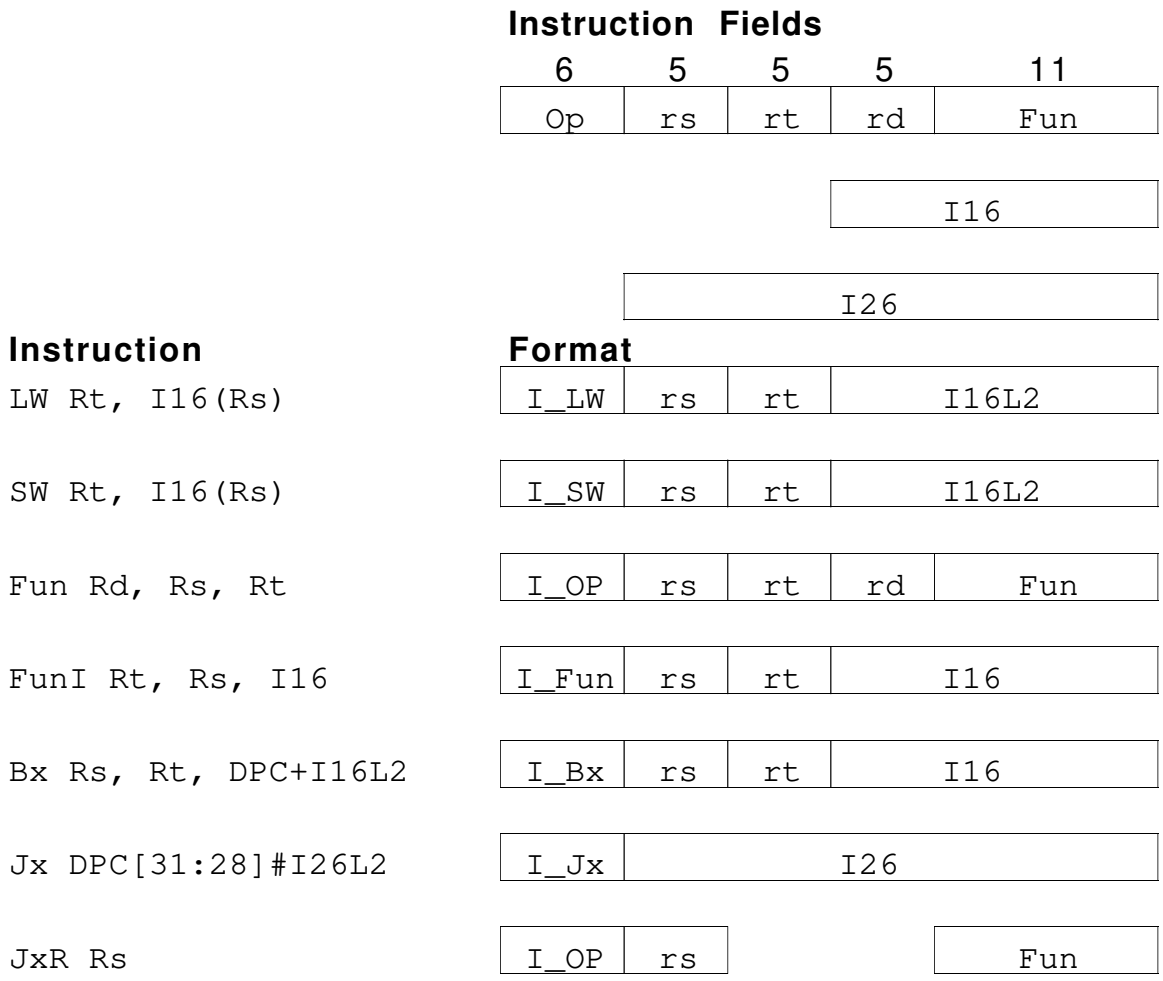
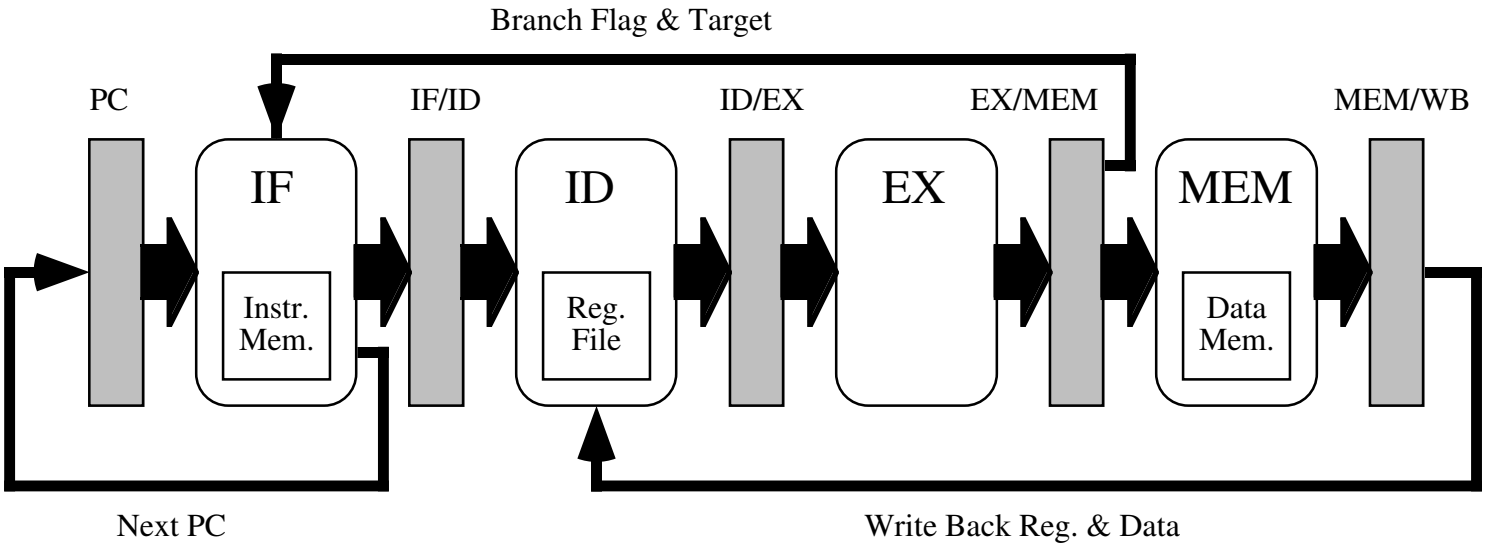
Notes on Assignment 2



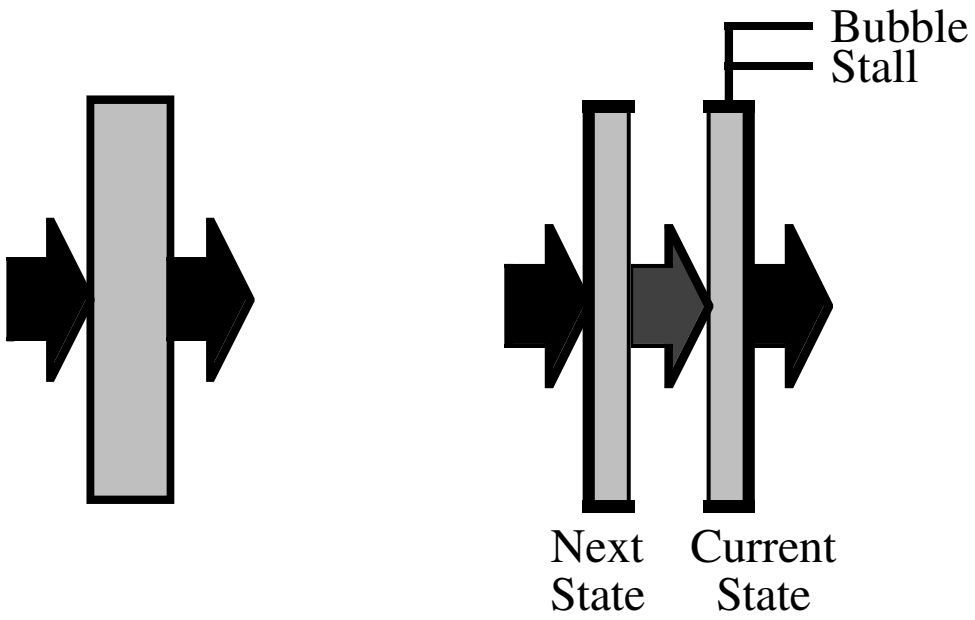
pMIPS Pipeline Structure



Notes

- **Each stage consists of operate logic connecting pipe registers**
- **WB logic merged into ID**
- **Additional paths required for forwarding**

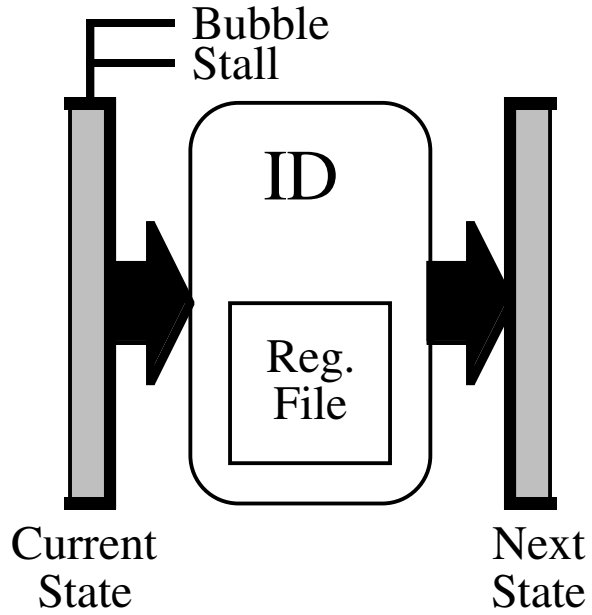
Pipe Register



Operation

- **Pipe registers updated after next states computed**
  - Load next state into current
- **Special update options**
  - *Stall*: retain current state
  - *Bubble*: set current state to all 0's

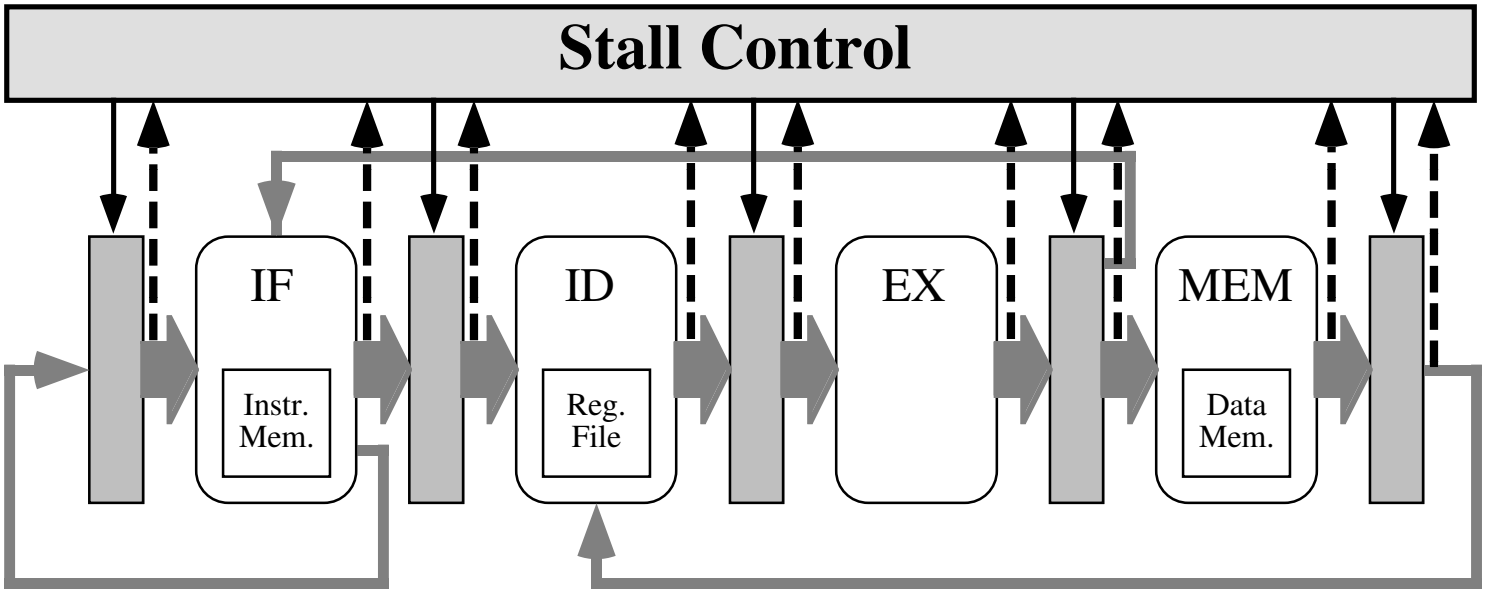
Pipeline Stage



Operation

- **Computes next state based on current**
  - From/to one or more pipe registers
- **May have embedded memory elements**
- **Signals encoded so that all 0's is like NOP**
  - When bubble generated

Stalling



Stall Logic

- **Determines which stages to stall or bubble on next update**
- **Based on pipe register inputs and outputs**
  - Assumes stages have already operated

Simulator Operation

```
/* Run pipeline for one cycle */
void step_pipe()
{
  /* Operate Phase */
  do_if_stage();
  do_id_stage();
  do_ex_stage();
  do_mem_stage();
  /* Stalls */
  do_stall_check();
  /* Update Phase */
  update_stages();
}
```

- **Each stage implemented as separate procedure**
- **Stall logic sets pipe register controls**
  - default is to update
- **Pipe registers updated**
  - `update_stages`

Pipe Register Allocation

Declaration

- **Define data type**
- **With appropriate fields**
- **Don't worry about bit packing**

```
/* Program counter */
typedef struct {
  unsigned pc;
} pc_ele, *pc_ptr;
```

```
/* IF/ID pipe register */
typedef struct {
  unsigned instr;
  unsigned incr_pc;
} if_id_ele, *if_id_ptr;
```

Creation

- **Request register of appropriate size**
- **Allocates storage for current and new states**

```
pc_state =
     new_stage(sizeof(pc_ele));
pc_next = pc_state->next;
pc_current = pc_state->current;
```

Stage Operation

```
void do_if_stage()
{
  unsigned pc = pc_current->pc;
  unsigned pc4 = pc+4;
  if_out->instr = instr_fetch(pc);
  if_out->incr_pc = pc4;
  /* Update PC to be either
     branch target or incremented PC */
  pc_next->pc = mem_in->branch_flag ?
    mem_in->branch_target : pc4;
}
```

Procedure for Stage

- **Computes stage outputs based on inputs**
- **Embedded memories / ALU implemented as procedures**
- **Multiplexors, decoding logic, etc. implemented with C expressions**

Stall Logic

```
if (ex_in->instr == BREAK_INSTR(0)) {
  break_encountered = 1;
  stall_ex = 1;
}

if (stall_ex) {
  pc_state->stall = 1;
  if_id_state->stall = 1;
  id_ex_state->stall = 1;
  /* Insert bubble into MEM */
  ex_mem_state->bubble = 1;
}
```

Stall Check

- **Determines which registers to stall or bubble on next update**
- **Currently only for BREAK instruction**
  - Detect in EX stage
  - Will continue to stall until end of simulation

Generating “Object” Code

```
/* Test of assembly code jumps */
#include "mips.h"
#include "gen_code.h"

code_word code[] = {
{ADDIU_INSTR(1,0,28), "0: addiu $1,$0,28 --> 28"},
{JALR_INSTR(1), "4:jalr $1 (to 20)"},
/* Delay slot */
{ADDIU_INSTR(2,31,0), "8: addiu $2,$31,0 --> 8"},

. . . .

/* Shouldn't hit this */
{BREAK_INSTR(0), "40: break skipped"},
/* Jump target */
{ADDIU_INSTR(4,4,4), "[j tgt]44: addiu $4,$4,4 --> 7"},
END_INSTR
};
```

- **Create initialized array**
- **Use macro definition of ISA**
- **Include text as annotation**

Object Code Format

```
0:0x2401001C   0: addiu $1,$0,28 --> 28
1:0x00200009   4:jalr $1 (to 20)
2:0x27E20000   8: addiu $2,$31,0 --> 8
3:0x24630004   [rtn tgt]12: addiu $3,$3,4 --> 4
4:0x0800000B   16:j 4*11
5:0x24630003   20: addiu $3,$3,3 --> 7
6:0x0000000D   24: break skipped
7:0x24040001   [jalr tgt]28: addiu $4,$0,1 --> 1
8:0x03E00008   32: jr $31
9:0x24840002   36: addiu $4,$4,2 --> 3
A:0x0000000D   40: break skipped
B:0x24840004   [j tgt]44: addiu $4,$4,4 --> 7
C:0x0000000D   break
D:0x00000000   nop
E:0x00000000   nop
```

- **One instruction per line**
- **Address and code word in hex**
- **Annotations at end of line**

Simulator Execution

```
Fetching instruction 0:0x2401001c 0: addiu $1,$0,28 --> 28
EX Results: ALU:0x0, BR:0, MW:0, MR:0, WB:0x0, B:0x0
Nonzero register state after 1 cycles
Fetching instruction 1:0x200009 4:jalr $1 (to 20)
EX Results: ALU:0x0, BR:0, MW:0, MR:0, WB:0x0, B:0x0
Nonzero register state after 2 cycles
Fetching instruction 2:0x27e20000 8: addiu $2,$31,0 --> 8
EX Results: ALU:0x1c, BR:0, MW:0, MR:0, WB:0x1, B:0x0
Nonzero register state after 3 cycles
Fetching instruction 3:0x24630004 [rtn tgt]12: addiu $3,$3,4 --> 4
Forwarded EX data 0x1c to EX, rs = 0x1
J(AL)R to 0x1c, storing 0xc in 0x1f
EX Results: ALU:0xc, BR:1, MW:0, MR:0, WB:0x1f, B:0x0
Nonzero register state after 4 cycles
Fetching instruction 4:0x800000b 16:j 4*11
Forwarded EX data 0xc to EX, rs = 0x1f
EX Results: ALU:0xc, BR:0, MW:0, MR:0, WB:0x2, B:0x0
Nonzero register state after 5 cycles
  01:0x0000001C
```