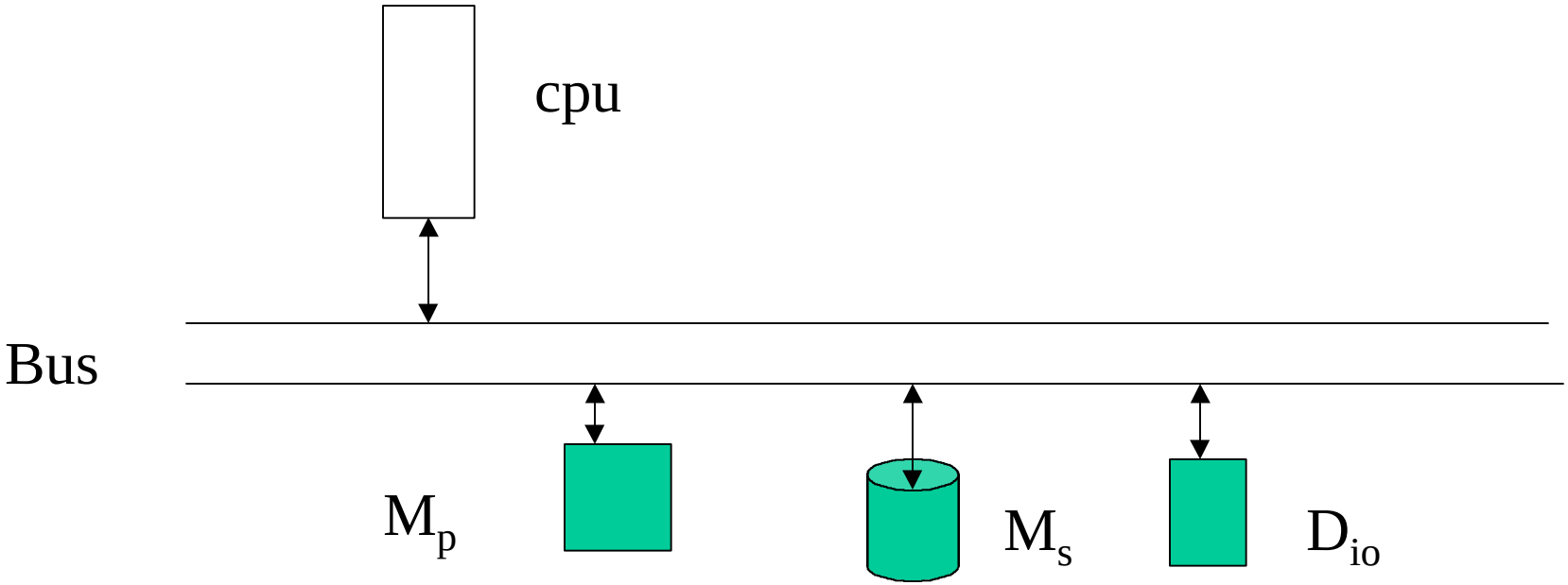
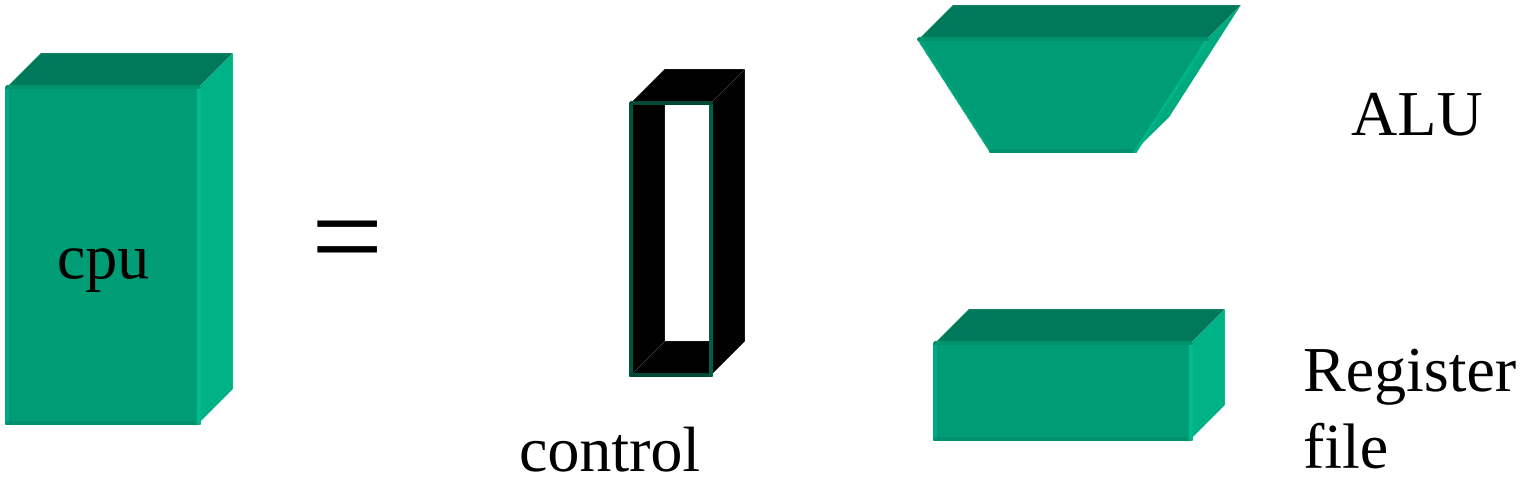
Review:
A simple architecture

- Single -bus
- slow simple cheap

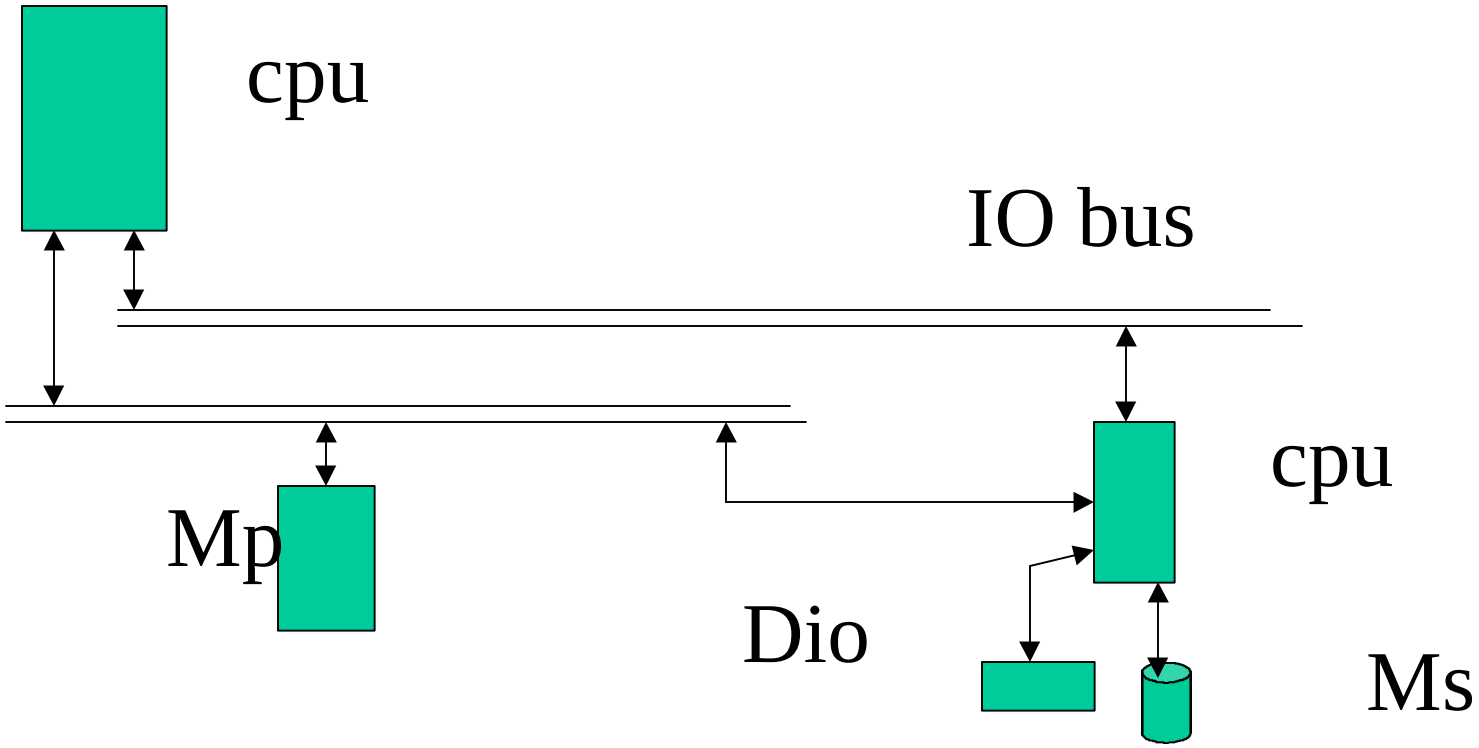


Review:
A simple architecture

- where



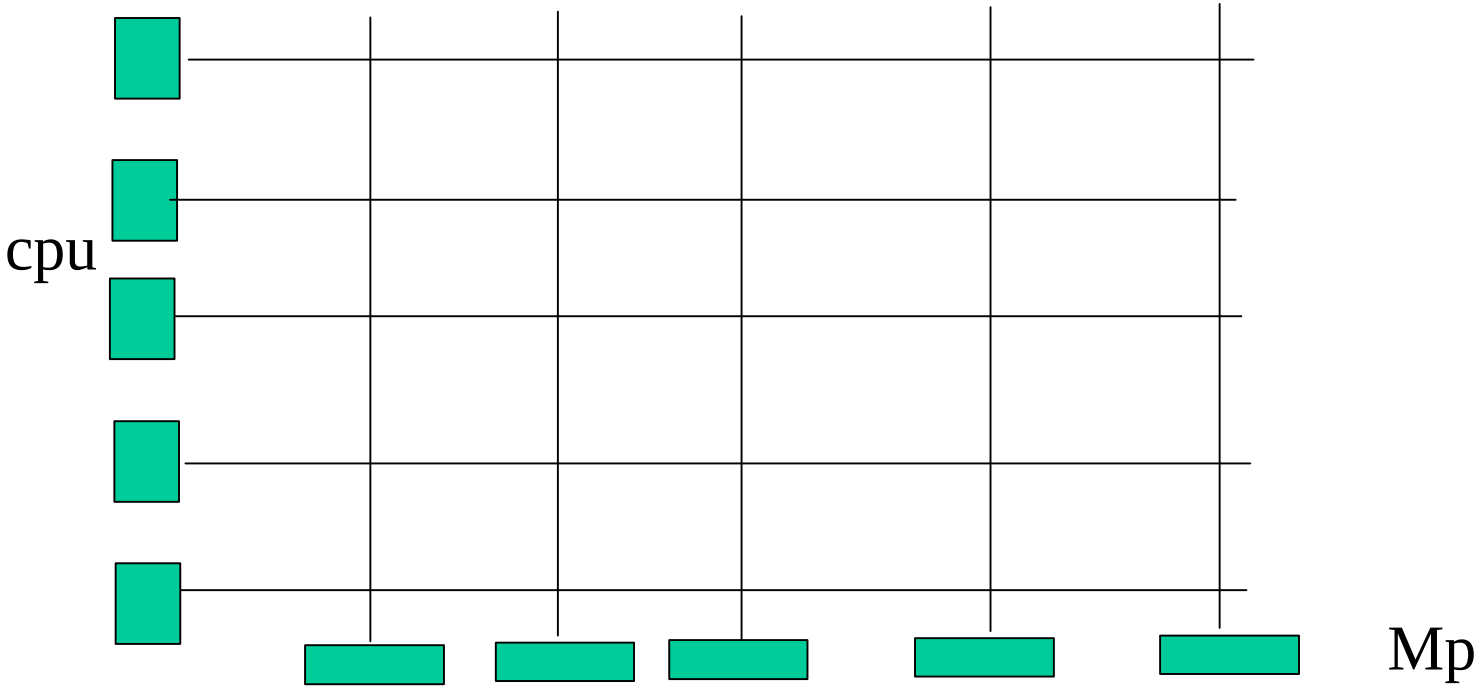
Slightly fancier architecture



Fancier PMS architecture

- Compare it to the simple one:
  - faster ? (why??)
  - more expensive?

Even fancier . . .
Crossbar-switched multiprocessor



Review:
what the cpu does (English)

- 1. Fetch next sequential instruction into Instruction Register (IR)
- 2. Update Program Counter (PC) to point to next instruction
- 3. If data operand required from Mp, calculate its address
- 4. Fetch operand(s) from Mp to cpu registers
- 5. Execute instruction
- 6. Store result(s)
- 7. Goto step 1.

Review:
what the cpu does (PL/1)

- INTERP: PROC (MEMORY, START_ADDR);
- /* Mp is word-addressed. Machine stores 1 instruction per word. Mp cells are named 0,1, . . . , 4095. There is 1 register called AC. INTERP halts when RUN = 0. Process state = {memory, PC, RUN, AC }.
- DCL (MEMORY(0:4095), AC, START_ADDR) FIXED BIN;
- DCL (PC, IR, RUN, TYPE, DATA, DATA_LOCN) FIXED BIN;
- PC = SSTART_ADDR;
- RUN = 1;

Review:
what the cpu does (PL/1)

- DO WHILE (RUN = 1);
- /* GET NEXT INSTRUCTION - FETCH PHASE */
- IR = MEMORY(PC);
- /* BUMP PC TO POINT TO NEXT INSTR */
- PC = PC + 1;
- /* DECODE INSTR & PUT ITS TYPE INTO TYPE */
- TYPE = DECODE (IR);
- /* LOCATE DATA USED BY INSTR. RETURNS -1 IF NONE */

Ctd...

- DATA_LOCN = FIND_DATA(IR);

- IF (DATA_LOCN > 0) THEN
  - DATA = MEMORY(DATA_LOCN);
- /* EXECUTE */
- CALL EXECUTE (TYPE,DATA, MEMORY, AC, PC, RUN);
- END;
- RETURN;
- END INTERP;

comments

- THIS REPRESENTATION IS
  - – *FUNCTIONALLY* ACCURATE
    - IT FAITHFULLY REFLECTS THE FUNCTIONING OF THE REAL CPU, BUT
  - – *STRUCTURALLY* INACCURATE
    - ITS SOFTWARE MODULES ARE NOT ISOMORPHIC TO THE HARDWARE MODULES OF THE REAL CPU

- ASSIGNMENT: REWRITE IT IN C

Compilers vs. Interpreters

- Def'n: A compiler analyzes *all* statements of a program, *then* executes all of them
  - AAAAAA EEEEEE
- Def'n: An interpreter analyzes each statement then executes it
  - AEAEAEAEAEAE
- Def'n: An *incremental compiler* . . .

Compilers vs. Interpreters

- Remark: a cpu is an interpreter, where the unit of execution is one instruction (Level 2)

- Remark: *some* cpus have a second, inner interpreter where the unit of execution is one microinstruction (Level 1)

- Remark: a cpu can interpretively execute the instruction set of another cpu (Level 4)

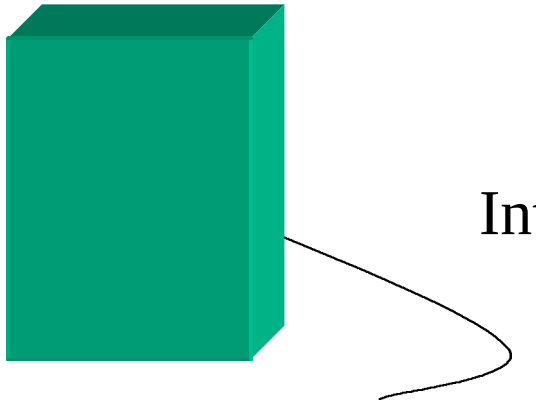
Review : Interrupts

- The problem:
  - get cpu to do something (ie execute some routine)
  - *right now* (ie subject to a hard realtime constraint)

- Why care?
  - Air traffic control
  - nuclear reactor control

Review : Interrupts

- Solution:

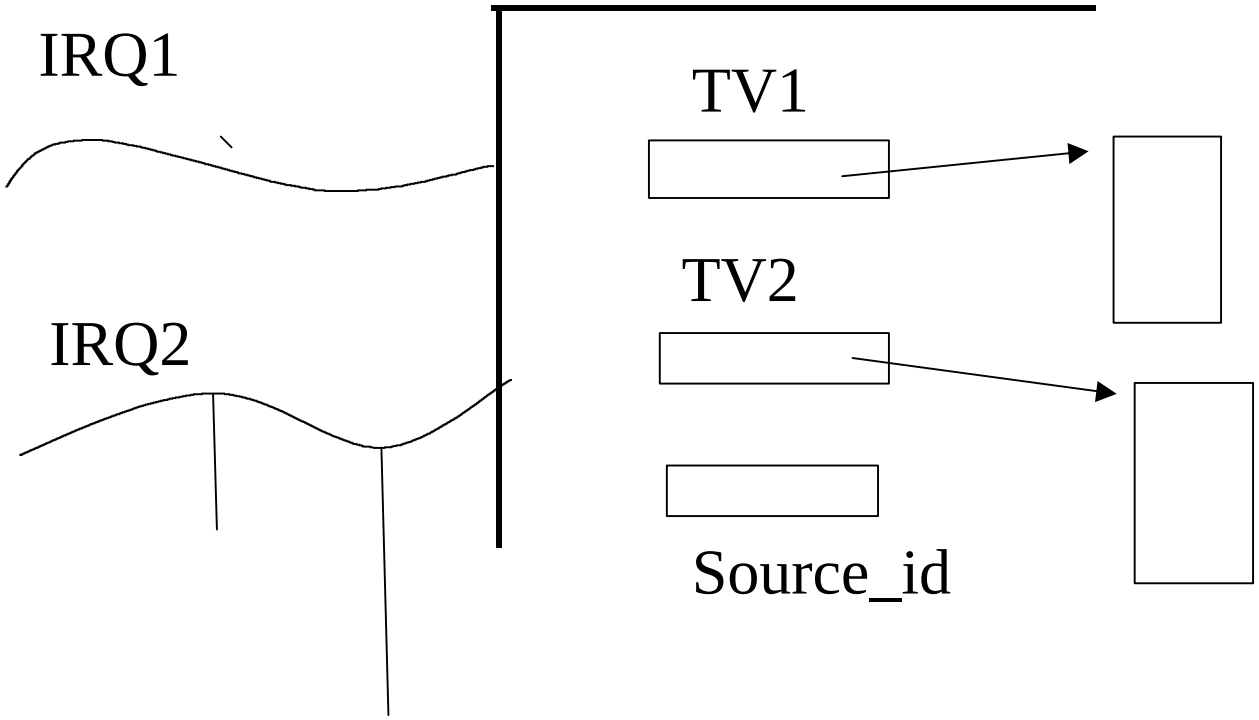


When (IRQ = = 1)
do save PS, PC
SAVE REGISTERS
C(C(XFER_VECTOR) -> PC
od

Review : Interrupts

- Embellishments:
  - mask bit
  - multiple lines, service routine for each
  - source identifier per line
    - eliminates polling the devices of the line
      - *vectored interrupts*
  - priorities among lines
    - usually external interrupts first, then
    - internal interrupts (traps) second

Illustrating ...



Sample Interrupt Structure- Motorola 68000 family

- Fully vectored by priority & device identifier
  - service routine never has to figure out who invoked it
- 8 levels of external priority, plus cpu priority (traps)
- cpu state saved (on a stack, natch) by hardware

M 68000 performance

Exception	cycles req'd	microsec
Address err	24	3
bus err	24	3
external interrupt	21	2.6
illegal instruction	16	2
privileged instruction	16	2
trace	16	

IBM System/360
Interrupt structure

- FIXED address vectors (Boo)
- NO interrupt stacking (nested interrupts) BOO!
- State saving:
  - PSW stored at FIXED address
  - no other cpu state info saved
- Interrupts maskable

IBM S/360

- Processor state:
  - formally organized as Processor Status Word (PSW)
  - note what ISN'T there e.g. general purpose register contents

bits	contents
0-7	interrupt masks: external, channels
8-11	protection key
12	ASCII mode
13	machine check mode

S/360 interrupts

- 14 wait state
- 15 problem state
- 16-31 interrupt id
- 32-33 instruction length code
- 34-35 condition code
- 36-39 trap mask
  - (fixed point overflow, floating point overflow, decimal overflow, exponent underflow, significance)
- 40-63 program counter

S/360 Interrupt Service routine

```
IOOPSW    EQU   56       OLD PSW
IONPSW    EQU   120      NEW PSW
…
              MVC   IONPSW(8), INTPSW(8)   TO-FROM
              {MAIN PROGRAM}
INTPSW    DC      X'0004 0000', A(INTER)
*** Interrupt service ROUTINE  ****
INTER       DS      0H
        {SAVE REGISTERS }
```

S/360 interrupt svce

```
CLC   IOOPSW+2(2), =x'000C'
                READER?
BE    READER
CLC   IOOPSW+2(2), =x'000D'
                PUNCH?
BE    PUNCH
B     ERROR
```

S/360 INTERRUPT SVCE

- READER {READER SERVICE}
- B END
- PUNCH {PUNCH SERVICE}
- B END
- END [RESTORE GPRS}
- LPSW IOOPSW RETURN TO INTERRUPTED PROGRAM

S/360 Permanent Storage Assignments

Addr	len	purpose
0	8	Initial PSW load
8	8	Initial load CCW1
16	8	Initial load CCW2
24	8	Ext. old PSW
32	8	Spv. Call old PSW
40	8	Pgmr old PSW
48	8	Machine check old PSW
56	8	I/O old PSW

S/360 Permanent Storage Assignments

• 88	8	external new PSW
• 96	8	Spvr call new PSW
• 104	8	Pgm new PSw
• 112	8	Machine check new PSW
• 120	8	I/O new PSW

- so , . . .

I/O Interrupt occurs

- Finish current instruction execution
- store current PSW at I/O old PSW (56)
- load PSW from I/O new PSW (120)
- fetch next instruction as directed by
  - c(PC) = part of c(PSW) -- the Interrupt Service Routine

PH Break
Interrupts & Exceptions (traps)

- You are responsible for
  - Section 5.6
- key concepts:
  - the MIPS cpu version of interrupt and exception handling

Summary

- A cpu is an interpreter
  - of instructions
  - implemented in hardware
- A cpu can simulate another cpu
- Instructions of conventional machine level (L2) may infact be interpreted by another interpreter running at L1 - the microprogram level
- hence *some* cpus (not all) are 2 nested interpreters