



COMPUTER ORGANIZATION AND DESIGN
 The Hardware/Software Interface



Chapter 2

Linking and Comparison

These slides are based on the slides by the authors. The slides doesn't include all the material covered in the lecture. The slides will be explained, modified, and sometime corrected in the lecture.



Chapter 2 — Instructions: Language of the Computer — 2

Synchronization

- Two processors sharing an area of memory
 - P1 writes, then P2 reads
 - Data race if P1 and P2 don't synchronize
 - Result depends of order of accesses
- Hardware support required
 - Atomic read/write memory operation
 - No other access to the location allowed between the read and write
- Could be a single instruction
 - E.g., atomic swap of register ↔ memory
 - Or an atomic pair of instructions

§2.11 Parallelism and Instructions: Synchronization

Synchronization in RISC-V

- Load reserved: `l r. d rd, (rs1)`
 - Load from address in rs1 to rd
 - Place reservation on memory address
- Store conditional: `sc. d rd, (rs1), rs2`
 - Store from rs2 to address in rs1
 - Succeeds if location not changed since the `l r. d`
 - Returns 0 in rd
 - Fails if location is changed
 - Returns non-zero value in rd



Synchronization in RISC-V

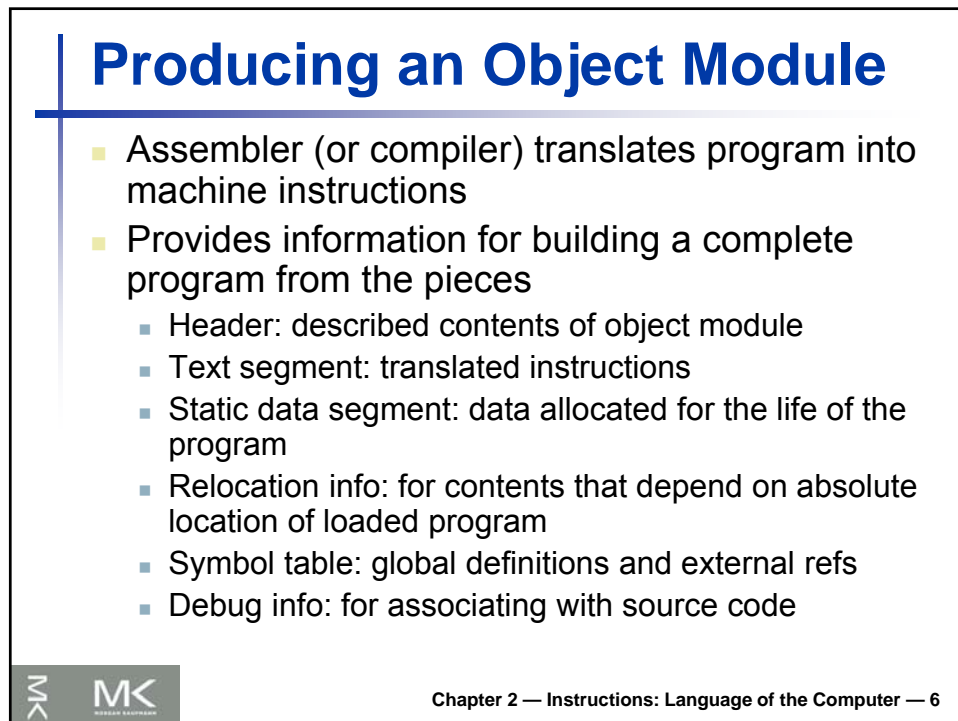
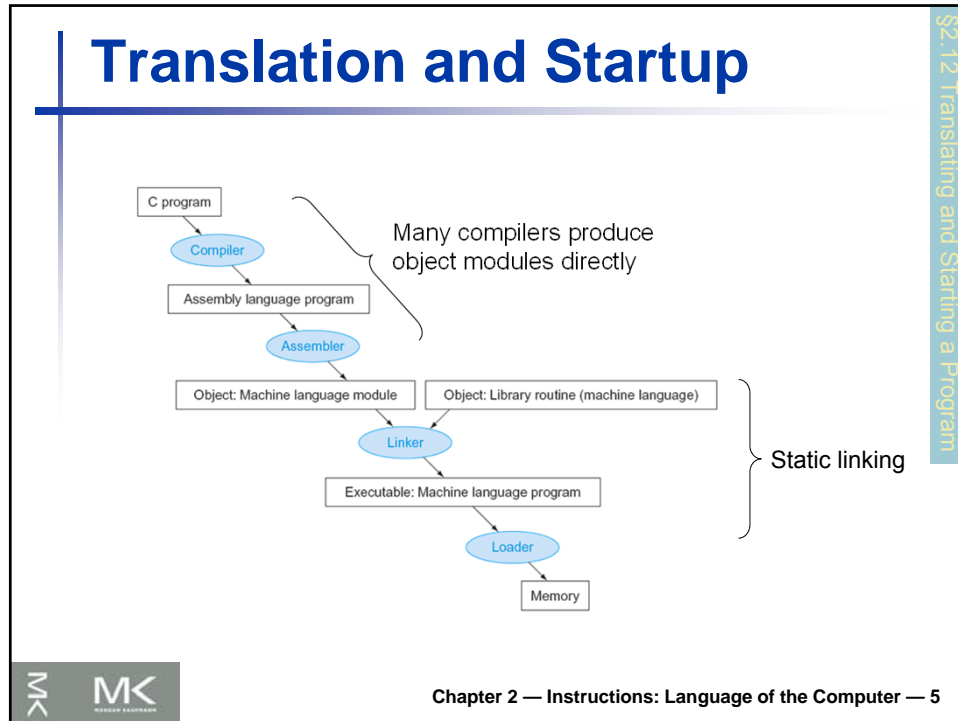
- Example 1: atomic swap (to test/set lock variable)


```
again: l r. d x10, (x20)
       sc. d x11, (x20), x23 // X11 = status
       bne x11, x0, again // branch if store failed
       addi x23, x10, 0 // X23 = loaded value
```
- Example 2: lock


```
addi x12, x0, 1 // copy locked value
again: l r. d x10, (x20) // read lock
       bne x10, x0, again // check if it is 0 yet
       sc. d x11, (x20), x12 // attempt to store
       bne x11, x0, again // branch if fails
```
- Unlock:


```
sd x0, 0(x20) // free lock
```





Linking Object Modules

- Produces an executable image
 1. Merges segments
 2. Resolve labels (determine their addresses)
 3. Patch location-dependent and external refs
- Could leave location dependencies for fixing by a relocating loader
 - But with virtual memory, no need to do this
 - Program can be loaded into absolute location in virtual memory space



Loading a Program

- Load from image file on disk into memory
 1. Read header to determine segment sizes
 2. Create virtual address space
 3. Copy text and initialized data into memory
 - Or set page table entries so they can be faulted in
 4. Set up arguments on stack
 5. Initialize registers (including sp, fp, gp)
 6. Jump to startup routine
 - Copies arguments to x10, ... and calls main
 - When main returns, do exit syscall

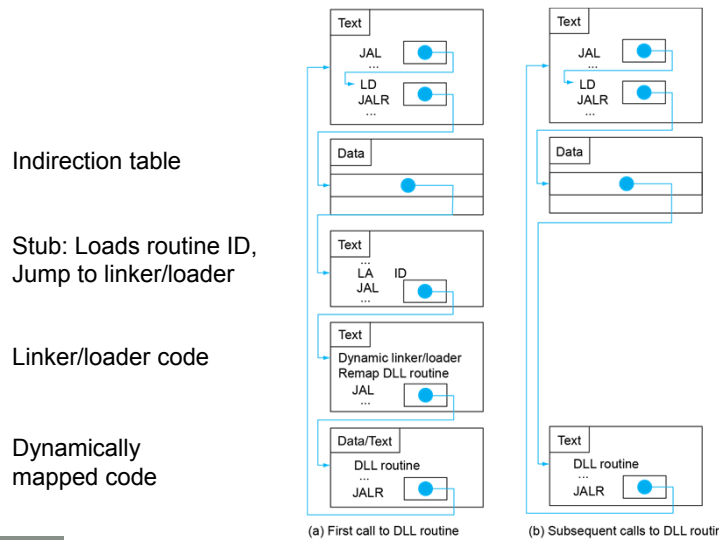


Dynamic Linking

- Only link/load library procedure when it is called
 - Requires procedure code to be relocatable
 - Avoids image bloat caused by static linking of all (transitively) referenced libraries
 - Automatically picks up new library versions

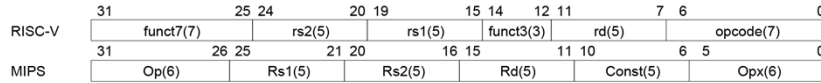


Lazy Linkage

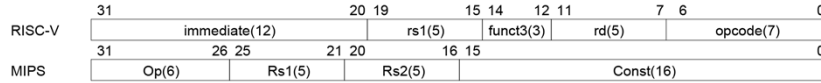


Instruction Encoding

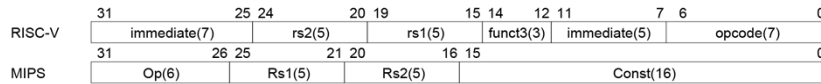
Register-register



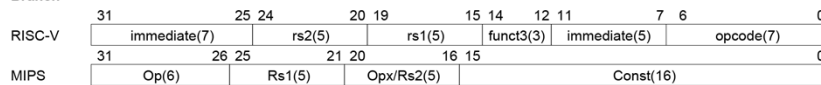
Load



Store



Branch



The Intel x86 ISA

§2.17 Real Stuff: x86 Instructions

- Evolution with backward compatibility
 - 8080 (1974): 8-bit microprocessor
 - Accumulator, plus 3 index-register pairs
 - 8086 (1978): 16-bit extension to 8080
 - Complex instruction set (CISC)
 - 8087 (1980): floating-point coprocessor
 - Adds FP instructions and register stack
 - 80286 (1982): 24-bit addresses, MMU
 - Segmented memory mapping and protection
 - 80386 (1985): 32-bit extension (now IA-32)
 - Additional addressing modes and operations
 - Paged memory mapping as well as segments



The Intel x86 ISA

- Further evolution...
 - i486 (1989): pipelined, on-chip caches and FPU
 - Compatible competitors: AMD, Cyrix, ...
 - Pentium (1993): superscalar, 64-bit datapath
 - Later versions added MMX (Multi-Media eXtension) instructions
 - The infamous FDIV bug
 - Pentium Pro (1995), Pentium II (1997)
 - New microarchitecture (see Colwell, *The Pentium Chronicles*)
 - Pentium III (1999)
 - Added SSE (Streaming SIMD Extensions) and associated registers
 - Pentium 4 (2001)
 - New microarchitecture
 - Added SSE2 instructions



Chapter 2 — Instructions: Language of the Computer — 15

The Intel x86 ISA

- And further...
 - AMD64 (2003): extended architecture to 64 bits
 - EM64T – Extended Memory 64 Technology (2004)
 - AMD64 adopted by Intel (with refinements)
 - Added SSE3 instructions
 - Intel Core (2006)
 - Added SSE4 instructions, virtual machine support
 - AMD64 (announced 2007): SSE5 instructions
 - Intel declined to follow, instead...
 - Advanced Vector Extension (announced 2008)
 - Longer SSE registers, more instructions
- If Intel didn't extend with compatibility, its competitors would!
 - Technical elegance ≠ market success



Chapter 2 — Instructions: Language of the Computer — 16

Basic x86 Registers

Name	Use
EAX	GPR 0
ECX	GPR 1
EDX	GPR 2
EBX	GPR 3
ESP	GPR 4
EBP	GPR 5
ESI	GPR 6
EDI	GPR 7
CS	Code segment pointer
SS	Stack segment pointer (top of stack)
DS	Data segment pointer 0
ES	Data segment pointer 1
FS	Data segment pointer 2
GS	Data segment pointer 3
EIP	Instruction pointer (PC)
EFLAGS	Condition codes

Chapter 2 — Instructions: Language of the Computer — 17

Basic x86 Addressing Modes

- Two operands per instruction

Source/dest operand	Second source operand
Register	Register
Register	Immediate
Register	Memory
Memory	Register
Memory	Immediate
- Memory addressing modes
 - Address in register
 - Address = $R_{base} + displacement$
 - Address = $R_{base} + 2^{scale} \times R_{index}$ (scale = 0, 1, 2, or 3)
 - Address = $R_{base} + 2^{scale} \times R_{index} + displacement$

Chapter 2 — Instructions: Language of the Computer — 18

x86 Instruction Encoding

a. JE EIP + displacement

4	4	8
JE	Condi- tion	Displacement

b. CALL

8	32
CALL	Offset

c. MOV EBX, [EDI + 45]

6	1	1	8	8
MOV	d	w	r/m Postbyte	Displacement

d. PUSH ESI

5	3
PUSH	Reg

e. ADD EAX, #6765

4	3	1	32
ADD	Reg	w	Immediate

f. TEST EDX, #42

7	1	8	32
TEST	w	Postbyte	Immediate

- Variable length encoding
 - Postfix bytes specify addressing mode
 - Prefix bytes modify operation
 - Operand length, repetition, locking, ...

Chapter 2 — Instructions: Language of the Computer — 19

Implementing IA-32

- Complex instruction set makes implementation difficult
 - Hardware translates instructions to simpler microoperations
 - Simple instructions: 1–1
 - Complex instructions: 1–many
 - Microengine similar to RISC
 - Market share makes this economically viable
- Comparable performance to RISC
 - Compilers avoid complex instructions

Chapter 2 — Instructions: Language of the Computer — 20

Other RISC-V Instructions

§2.18 The Rest of the RISC-V Instruction Set

- Base integer instructions (RV64I)
 - Those previously described, plus
 - `auipc rd, imm` // $rd = (imm \ll 12) + pc$
 - follow by `jalc` (adds 12-bit `imm`) for long jump
 - `slt`, `sltu`, `slti`, `sltui`: set less than (like MIPS)
 - `addw`, `subw`, `addiw`: 32-bit add/sub
 - `sllw`, `srlw`, `srlw`, `slliw`, `srliw`, `sraiw`: 32-bit shift
- 32-bit variant: RV32I
 - registers are 32-bits wide, 32-bit operations



Chapter 2 — Instructions: Language of the Computer — 21

Instruction Set Extensions

- M: integer multiply, divide, remainder
- A: atomic memory operations
- F: single-precision floating point
- D: double-precision floating point
- C: compressed instructions
 - 16-bit encoding for frequently used instructions




Chapter 2 — Instructions: Language of the Computer — 22

Fallacies

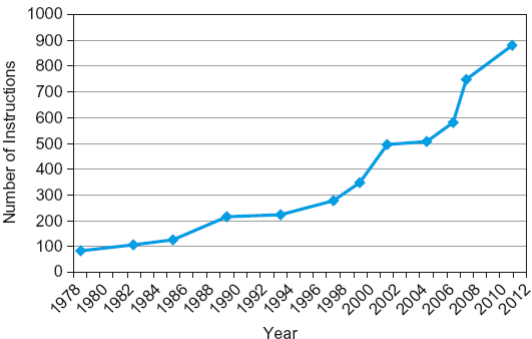
§2.19 Fallacies and Pitfalls

- Powerful instruction \Rightarrow higher performance
 - Fewer instructions required
 - But complex instructions are hard to implement
 - May slow down all instructions, including simple ones
 - Compilers are good at making fast code from simple instructions
- Use assembly code for high performance
 - But modern compilers are better at dealing with modern processors
 - More lines of code \Rightarrow more errors and less productivity


Chapter 2 — Instructions: Language of the Computer — 23


Fallacies

- Backward compatibility \Rightarrow instruction set doesn't change
 - But they do accrete more instructions



Year	Number of Instructions
1978	100
1980	110
1982	120
1984	130
1986	140
1988	180
1990	220
1992	230
1994	250
1996	280
1998	350
2000	450
2002	500
2004	520
2006	580
2008	750
2010	850
2012	900

x86 instruction set


Chapter 2 — Instructions: Language of the Computer — 24

Pitfalls

- Sequential words are not at sequential addresses
 - Increment by 4, not by 1!
- Keeping a pointer to an automatic variable after procedure returns
 - e.g., passing pointer back via an argument
 - Pointer becomes invalid when stack popped



Concluding Remarks

- Design principles
 1. Simplicity favors regularity
 2. Smaller is faster
 3. Good design demands good compromises
- Make the common case fast
- Layers of software/hardware
 - Compiler, assembler, hardware
- RISC-V: typical of RISC ISAs
 - c.f. x86

