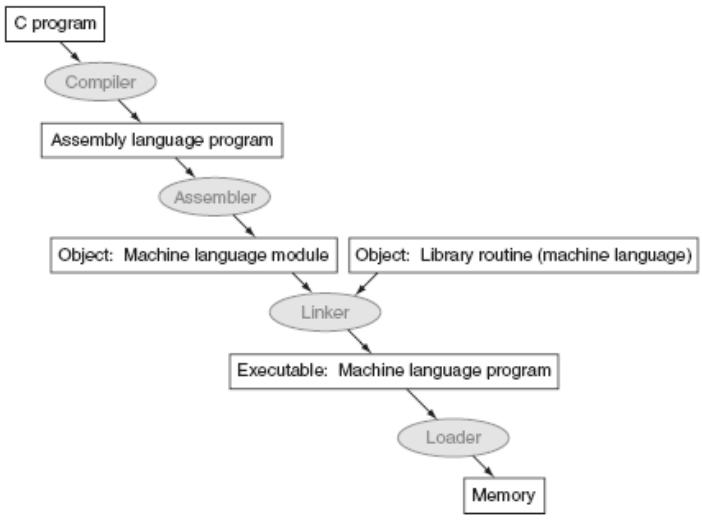
Lec 13: Linking and Memory

Kavita Bala
CS 3410, Fall 2008
Computer Science
Cornell University

Announcements

- PA 2 is out
  - Due on Oct 22nd
- Prelim
  - Oct 23rd, 7:30-9:30/10:00
  - All content up to Lecture on Oct 16th
- Ask us questions if in doubt

From Assembly to Running



Big Picture

- Assembler output is obj files
  - Not executable
  - May refer to external symbols
  - Each object file has its own address space
- Linker joins these object files into one executable
- Loader brings it into memory and executes

Object file

- Header
  - Size and position of pieces of file
- Text Segment
  - instructions
- Data Segment
  - Static data
- Relocation Information
  - Instructions and data that depend on absolute addresses
- Symbol Table
  - External and unresolved references
- Debugging Information

References

- Global labels
  - External: can be referenced from outside
  - In example
    - Main
- Local labels

Forward References

- Local labels can have forward references
- Two-pass assembly
  - Do a pass through the whole program, allocate instructions and lay out data, thus determining addresses
  - Do a second pass, emitting instructions and data, with the correct label offsets now determined

Forward References

- One-pass (or backpatch) assembly
  - Do a pass through the whole program, emitting instructions, emit a 0 for jumps to labels not yet determined, keep track of where these instructions are
  - Backpatch, fill in 0 offsets as labels are defined
- Pros and cons
  - Faster
  - But need to hold whole program in memory

Handling Forward References

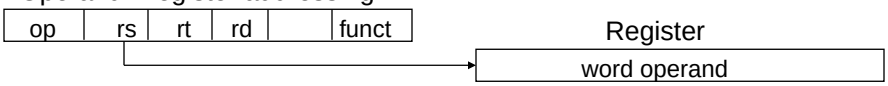
- Example:
  - bne $1, $2, L
    sll $0, $0, 0
    L: addiu $2, $3, 0x2
- The assembler will change this to
  - bne $1, $2, +1
    sll $0, $0, 0
    addiu $2, $3, 0x2

Handling Forward References

- Final machine code
  - 0X14220001 # bne
    0x00000000 # sll
    0x24620002 # addiu

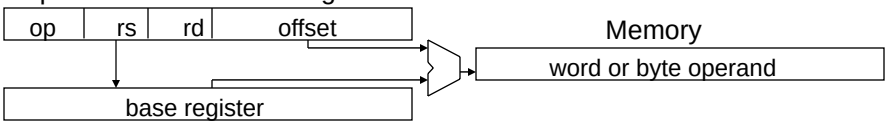
MIPS Addressing Modes

1. Operand: Register addressing



Eg: jr

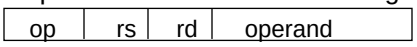
2. Operand: Base addressing



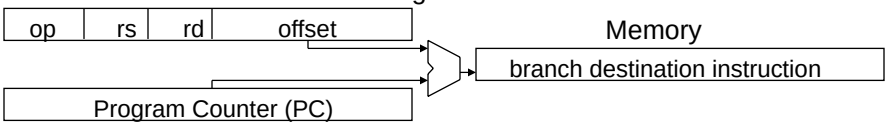
Eg: lw, sw

MIPS Addressing Modes

3. Operand: Immediate addressing

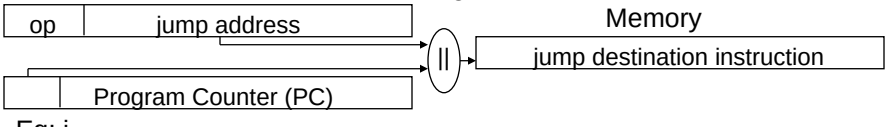


4. Instruction: PC-relative addressing



Eg: beq, bne

5. Instruction: Pseudo-direct addressing



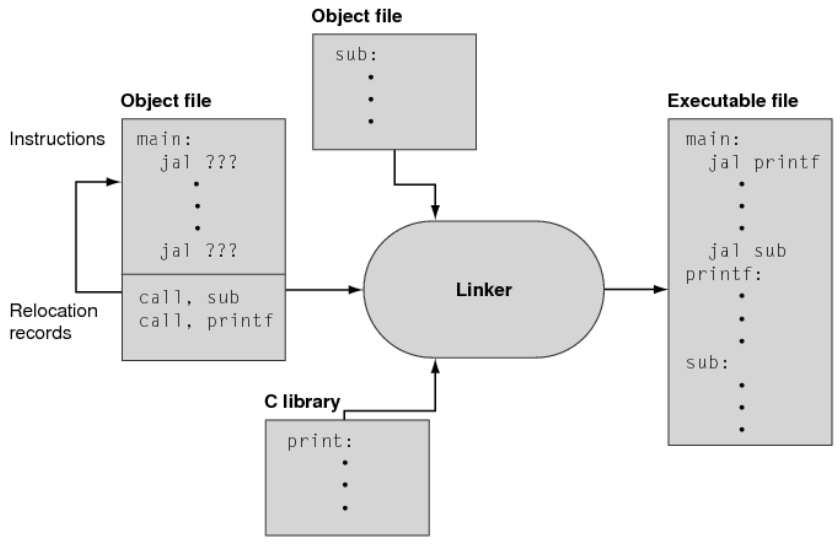
Eg: j

Separate Compilation

- Separately compiling modules and linking them together removes the need to recompile the whole program every time something changes

- Need to just recompile a small module
- A linker coalesces object files together to create a complete program

Linker



Linkers

- Combine object files into an executable
  - Resolve symbols
  - Creates final executable
  - Relocate if needed
  - Stores entry point in executable so processor knows where to start executing
- End result: a program on disk, ready to execute

Executable File Structure

- Header
- Text Segment
  - Instructions, ready to run as is
- Data Segment
  - Static data
- Symbol Table
  - Optional: for debugging

File Formats

- Unix
  - a.out format
  - COFF: Common Object File Format
  - ELF: Executable and Linking Format
  - All support executable and object files

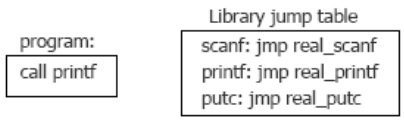
Libraries

- Collection of object files
- Linker adds all object files necessary to resolve undefined references
- User-specified order

Shared Libraries

- Libraries take too much memory for many applications
- Solution: share libraries

- Executable file
  - Points to library code (jump table)
  - Library compiled at fixed address



Linkers

- Static linkers
  - Doesn't reflect changes
  - Whole library (big)

- Dynamic linkers
  - Dynamically linked libraries (dlls)
  - Integrate code at runtime
    - One copy of shared library in memory
    - Load time cost

Dynamic Shared Objects

- Unix
  - Code compiled as dynamic shared object (DSO)
  - A relocatable shared library
  - Use ELF format
  - Supports Position Independent Code (PIC)
    - Program determines its current address
    - Add constant offset to access local data
    - If data located in a different library, use indirection through a Global Offset Table (GOT).
    - Address of GOT usually computed and stored in a register at the beginning of each procedure

Loaders

- Reads executable from disk
- Loads code and data into memory
- Initializes registers, stack, arguments
- Jumps to entry-point
- Part of the Operating System (OS)

Memory

Caches And Memory

- Chapter 7 (H & P)

Memory So Far

Big array of storage with more complex indexing than registers

- Addressing modes help us access memory
- A[i]; use base + displacement
- Use less space in instruction than immediate field

SRAM vs. DRAM

- SRAM (static random access memory)
  - Faster than DRAM
  - Each storage cell is larger (4-6 transistors)
  - So smaller capacity for same area
  - 2-10ns access time
- DRAM (dynamic random access memory)
  - Each storage cell tiny (capacitance on wire)
  - Can get 1-2GB chips
  - 50-150ns access time
  - Leaky–needs to refresh data periodically

Performance

- CPU clock rates ~0.2ns-2ns (5GHz-500MHz)

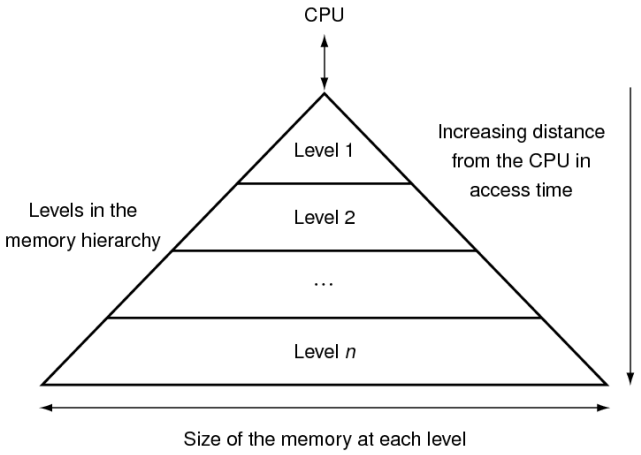
Technology	Capacity	Cost/GB	Latency
Tape	1 TB	$.17	100s
Disk	300 GB	$.34	Millions cycles (ms)
DRAM	4GB	$100s	50-300 cycles (10s of ns)
SRAM off	512KB	$4-10'sk	5-15 cycles (few ns)
SRAM on	16 KB	???	1-3 cycles (ns)

- Capacity and latency are closely coupled
- Cost is inversely proportional

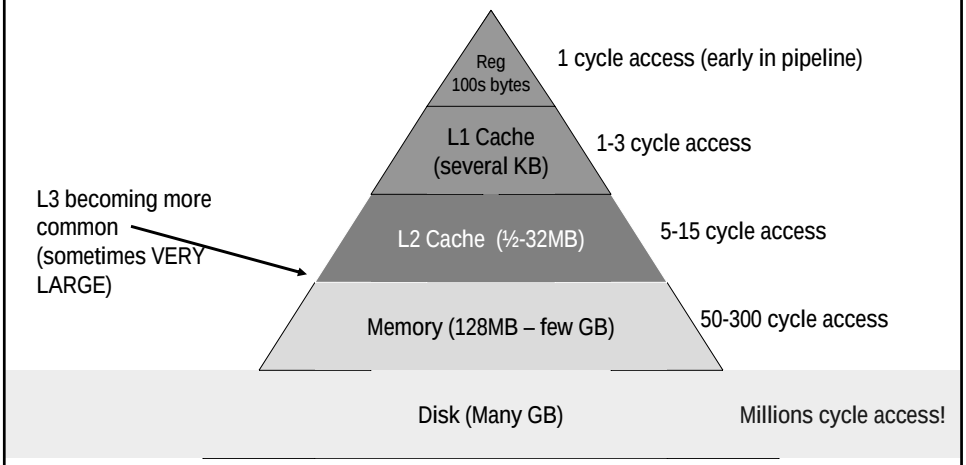
- How do we create the illusion of large and fast memory?

Memory Hierarchy

- Idea: Hide latency using small, fast memories called caches



Cache Design 101



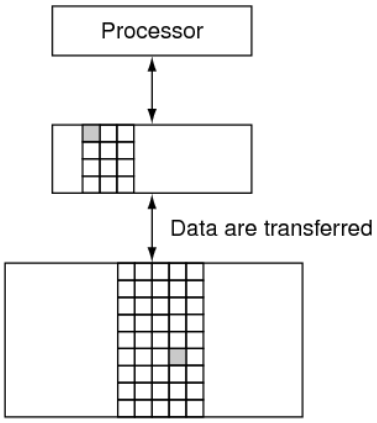
These are rough numbers: mileage may vary for latest/greatest
Caches USUALLY made of SRAM

Insight of Caches

- Exploit locality
  - Two types: temporal and spatial
- Temporal locality
  - If memory location X is accessed, then it is more likely to be accessed again in the near future than some random location Y
  - Caches exploit temporal locality by placing a memory element that has been referenced into the cache
- Spatial locality
  - If memory location X is accessed, then locations near X are more likely to be accessed in the near future than some random location Y
  - Caches exploit spatial locality by allocating a cache line of data (including data near the referenced location)

Memory Hierarchy

- Closer to processor
  - Subset of memory farther from processor
  - Faster and smaller
  - Transfer an entire block



Cache Lookups (Read)

- Look at address issued by processor
- Search cache to see if that block is in the cache
  - Hit: Block is in the cache
    - return requested data
  - Miss: Block is not in the cache
    - read line from memory
    - evict an existing line from the cache
    - place new line in cache
    - return requested data