Getting Started with Assembly Language

Slides revised 2/2/2014 by Patrick Kelley

Comparing ASM to High-Level Languages

Type of Application	High-Level Languages	Assembly Language
Business application software, written for single platform, medium to large size.	Formal structures make it easy to organize and maintain large sections of code.	Minimal formal structure, so one must be imposed by programmers who have varying levels of experience. This leads to difficulties maintaining existing code.
Hardware device driver.	Language may not provide for direct hardware access. Even if it does, awkward coding techniques must often be used, resulting in maintenance difficulties.	Hardware access is straightforward and simple. Easy to maintain when programs are short and well documented.
Business application written for multiple platforms (different operating systems).	Usually very portable. The source code can be recompiled on each target operating system with minimal changes.	Must be recoded separately for each platform, often using an assembler with a different syntax. Difficult to maintain.
Embedded systems and computer games requiring direct hardware access.	Produces too much executable code, and may not run efficiently.	Ideal, because the executable code is small and runs quickly.

Irvine, Kip R. Assembly Language for Intel-Based Computers 6/e, 2010.

Translating Languages

English: Display the sum of A times B plus C.

C++: cout << (A * B + C);

Assembly Language:

```
mov eax,A
mul B
add eax,C
call WriteInt
```

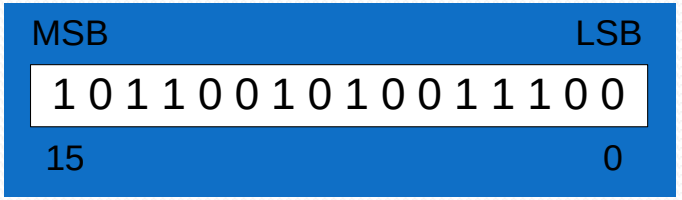

Intel Machine Language:

```
A1 00000000
F7 25 00000004
03 05 00000008
E8 00500000
```

Irvine, Kip R. Assembly Language for Intel-Based Computers 6/e, 2010.

Binary Numbers

- Digits are 1 and 0
  - 1 = true
  - 0 = false
- MSB – most significant bit
- LSB – least significant bit
- Bit numbering:



Integer Storage Sizes

Standard sizes:

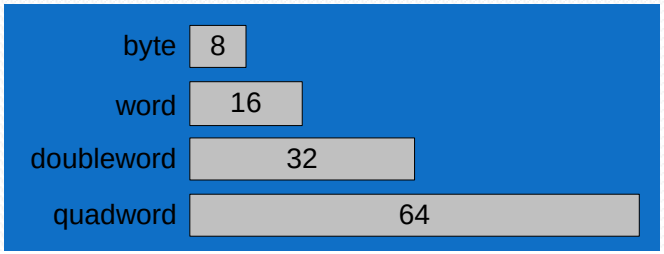


Table 1-4 Ranges of Unsigned Integers.

Storage Type	Range (low–high)	Powers of 2
Unsigned byte	0 to 255	0 to $(2^8 - 1)$
Unsigned word	0 to 65,535	0 to $(2^{16} - 1)$
Unsigned doubleword	0 to 4,294,967,295	0 to $(2^{32} - 1)$
Unsigned quadword	0 to 18,446,744,073,709,551,615	0 to $(2^{64} - 1)$

The above chart is for Intel x86 processors. In MIPS, a word is 32 bits, a half-word is 16 bits and there is no 'quadword'. A byte is still a byte.

What is the largest unsigned integer that may be stored in 20 bits?

Ranges of Signed Integers

The highest bit is reserved for the sign. This limits the range:

Storage Type	Range(low-high)	Powers of 2
Signed byte	-128 to +127	-2^7 to (2^7-1)
Signed half-word	-32,768 to +32767	-2^{15} to $(2^{15}-1)$
Signed word	-2,147,483,648 to + 2,147,483,647	-2^{31} to $(2^{31}-1)$

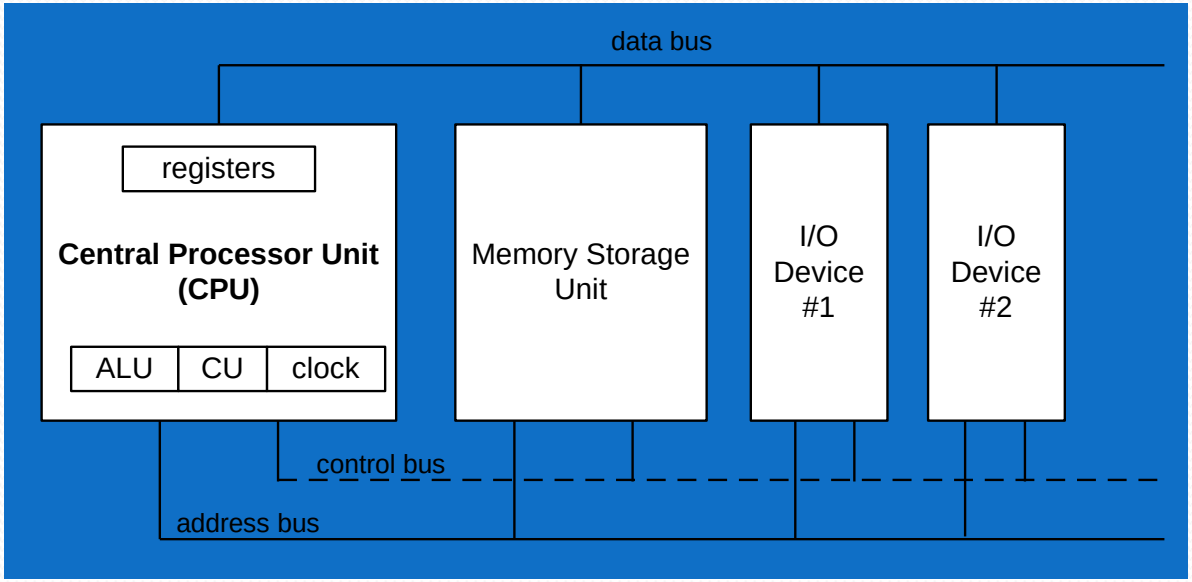
Practice: What is the largest positive value that may be stored in 20 bits?

Character Storage

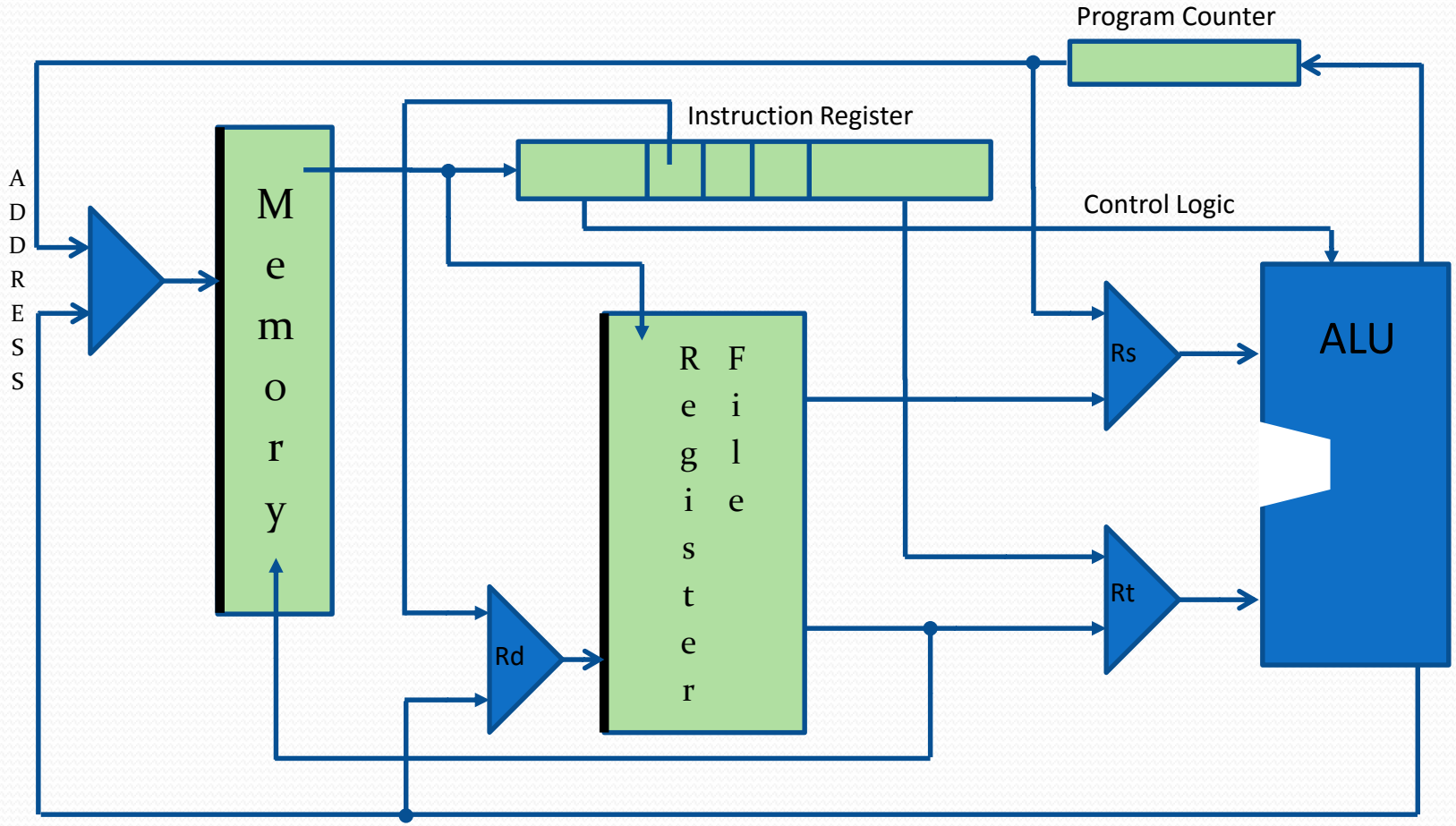
- Character sets
  - Standard ASCII (0 – 127)
  - Extended ASCII (0 – 255)
  - Unicode (0 – 65,535)
  - SPIM System IO only handles ASCII
- Null-terminated String
  - Array of characters followed by a *null byte*
- Using the ASCII table
  - Appendix B in MIPS book

Basic Microcomputer Design

- clock synchronizes CPU operations
- control unit (CU) coordinates sequence of execution steps
- ALU performs arithmetic and bitwise processing



MIPS Architechture



Instruction Execution Cycle

- Fetch
  - Memory at PC moved to IR
- Decode
  - Opcode to ALU
  - Register Operands selected
  - Immediate Operand to ALU
- Fetch operands
  - In other architectures but not MIPS
- Execute
- Store output
  - Register to Memory operation (MIPS)
  - PC Incremented
    - Instruction Addresses on 4-byte boundary
    - Branch may change PC instead of incrementing

Addressable Memory

- Addresses are 32 bits
  - 2^{32} locations = 4GB = 4,294,967,296 locations
  - The 'B' in GB stands for 'bytes'
  - Range is 0 to 2^{32}-1
- Half-word
  - 16 bits or 2 bytes long
  - Successive half-words have addresses that increment by 2
- Word
  - 32 bits or 4 bytes long
  - Successive words have addresses that increment by 4

Register ‘File’

- MIPS has 32 accessible 32-bit registers
  - The registers are numbered 0 to 31
  - They also have short names
  - The names will indicate the register’s purpose
  - Purpose may be by convention or design
- Special registers may not be user accessible
  - MIPS includes the IR, PC, and ALU registers
  - Sometimes special registers are read only
- Other architectures have additional registers
  - Commonly, there is a ‘status’ register
  - IO port registers are also common.

Register 'File' (continued)

Register	Number	Usage
zero	0	Constant 0
at	1	Reserved for assembler
v0	2	Used for return values from function calls
v1	3	
a0	4	Used to pass arguments to functions
a1	5	
a2	6	
a3	7	

Register 'File' (continued)

Register	Number	Usage
t0	8	Temporary Caller saved Called function need not save or protect these
t1	9	
t2	10	
t3	11	
t4	12	
t5	13	
t6	14	
t7	15	
t8	24	
t9	25	

Register 'File' (continued)

Register	Number	Usage
s0	16	Saved Temporary Callee saved Called function must protect these by saving and then restoring before returning to caller
s1	17	
s2	18	
s3	19	
s4	20	
s5	21	
s6	22	
s7	23	

Register ‘File’ (continued)

Register	Number	Usage
k0	26	Reserved for OS kernel
k1	27	
gp	28	Pointer to global area
sp	29	Stack Pointer
fp	30	Frame Pointer
ra	31	Return Address for function calls

Instruction Format

Register format

Op-code	Rs	Rt	Rd	unused	Function
000000	SSSSS	TTTTT	DDDDD	00000	FFFFFF

Immediate format

Op-code	Rs	Rt	Immediate constant
000000	SSSSS	TTTTT	IIIIIIIIIIIIIIII

Jump format

Op-code	Target
00001F	TTTTTTTTTTTTTTTTTTTTTTTTTT

Jump destination is PC + (Target Left Shift by 2)

Using Memory

- IR addresses memory directly via the PC
- MIPS ISA is 'load/store'
  - Typically, only a load or store instruction affects memory
  - All other operations affect only registers
  - Registers are used to point to memory for load/store
- Addressing mode is Base/Displacement
  - Displacement is also known as Offset
  - One register will hold a base address, another the displacement
  - Concatenating the two produces an effective address
- Other architectures use different modes
  - Direct addressing – specifies the location directly
  - Indirect addressing – part of the address is stored in memory
  - Indexed addressing – displacement is multiplied by a constant
  - Segment/offset – used to address large memory