Quick Reference: Assembler

Compilers & Tools
Compilers: Windows: Visual Studio Express, Cygwin, MINGW, Linux: GCC
Tools: Intel: Compiler, V-Tune, Parallel Studio, AMD: Open64 Compiler, CodeAnalyst

Registers
Note: GNU assembler uses '%' prefix to refer to registers.
Note: GNU assembler has destination last, Intel/ARM assembler has destination first

<table>
<thead>
<tr>
<th></th>
<th>Intel</th>
<th>ARM</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAX</td>
<td>32-bit</td>
<td>R0…R12</td>
</tr>
<tr>
<td>EAX</td>
<td>16-bit</td>
<td></td>
</tr>
<tr>
<td>AX</td>
<td>8-bit</td>
<td></td>
</tr>
<tr>
<td>AH, AL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RBX</td>
<td></td>
<td>R0…R12</td>
</tr>
<tr>
<td>EBX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BH, BL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RCX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ECX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CH, CL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RDX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EDX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DH, DL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RSI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ESI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RDI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EDI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R8…R15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RBP</td>
<td></td>
<td>SP=R13</td>
</tr>
<tr>
<td>EBP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RSP</td>
<td></td>
<td>LR=R14</td>
</tr>
<tr>
<td>ESP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RIP</td>
<td></td>
<td>PC=R15</td>
</tr>
<tr>
<td>EIP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IP</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Flags

<table>
<thead>
<tr>
<th></th>
<th>Intel</th>
<th>ARM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SF</td>
<td>sign (negative)</td>
<td>N</td>
</tr>
<tr>
<td>ZF</td>
<td>zero</td>
<td>Z</td>
</tr>
<tr>
<td>CF</td>
<td>carry</td>
<td>C</td>
</tr>
<tr>
<td>OF</td>
<td>overflow</td>
<td>V</td>
</tr>
</tbody>
</table>

Instruction Format

<table>
<thead>
<tr>
<th>Intel</th>
</tr>
</thead>
<tbody>
<tr>
<td>opcode</td>
</tr>
<tr>
<td>8-16 bits</td>
</tr>
</tbody>
</table>

Dr. Gordon Cichon  
gordon.cichon@ifi.lmu.de  
https://www.tcs.ifi.lmu.de/teaching/summer-2016/modern
Quick Reference: Assembler

ARM

<table>
<thead>
<tr>
<th>Intel</th>
<th>ARM</th>
<th>(encoding OP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>add</td>
<td>ADD 0100</td>
</tr>
<tr>
<td>ADC</td>
<td>add with carry</td>
<td>ADC 0101</td>
</tr>
<tr>
<td>SUB</td>
<td>subtract</td>
<td>SUB 0010</td>
</tr>
<tr>
<td>SBB</td>
<td>subtract with carry/borrow</td>
<td>SBC 0110</td>
</tr>
<tr>
<td></td>
<td>reverse subtract</td>
<td>RSB 0011</td>
</tr>
<tr>
<td></td>
<td>reverse subtract with carry</td>
<td>RSC 0111</td>
</tr>
<tr>
<td>NEG</td>
<td>negate</td>
<td></td>
</tr>
<tr>
<td>INC</td>
<td>increment</td>
<td></td>
</tr>
<tr>
<td>DEC</td>
<td>decrement</td>
<td></td>
</tr>
<tr>
<td>CMP</td>
<td>compare (subtraction w/ flags only)</td>
<td>CMP 1010</td>
</tr>
<tr>
<td></td>
<td>compare negated (addition w/ flags only)</td>
<td>CMPN 1011</td>
</tr>
</tbody>
</table>

Table 1: ARM Operation Encoding

Table 2: ARM Shift Encoding

1. Data Processing Operations

1.1 Arithmetic Operations
## Quick Reference: Assembler

### 1.2 Logic Operations

<table>
<thead>
<tr>
<th>Intel</th>
<th>ARM</th>
<th>(encoding OP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>bitwise ∧</td>
<td>AND 0000</td>
</tr>
<tr>
<td>OR</td>
<td>bitwise ∨</td>
<td>ORR 1100</td>
</tr>
<tr>
<td>NOT</td>
<td>bitwise ¬</td>
<td>MOVN 1111</td>
</tr>
<tr>
<td></td>
<td>move operation</td>
<td>MOV 1101</td>
</tr>
<tr>
<td>XOR</td>
<td>bitwise ⊕</td>
<td>MOVN 1111</td>
</tr>
<tr>
<td>TEST</td>
<td>test (bitwise ∧ w/ flags only)</td>
<td>TST 1000</td>
</tr>
<tr>
<td></td>
<td>test equal (bitwise ⊕ w/ flags only)</td>
<td>TEQ 1001</td>
</tr>
<tr>
<td>BT</td>
<td>bit test</td>
<td>BIC 1110</td>
</tr>
</tbody>
</table>

### 1.3 Shift Operations

<table>
<thead>
<tr>
<th>Intel</th>
<th>ARM</th>
<th>(encoding ST)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHL</td>
<td>logical shift left</td>
<td>LSL 00</td>
</tr>
<tr>
<td>SHR</td>
<td>logical shift right</td>
<td>LSR 01</td>
</tr>
<tr>
<td>SAR</td>
<td>arithmetic shift right</td>
<td>ASR 10</td>
</tr>
<tr>
<td>ROR</td>
<td>rotate right</td>
<td>ROR 11</td>
</tr>
<tr>
<td>ROL</td>
<td>rotate left</td>
<td></td>
</tr>
<tr>
<td>RCL</td>
<td>rotate right through carry</td>
<td></td>
</tr>
<tr>
<td>RCR</td>
<td>rotate right through carry</td>
<td>RRX 11 (with SHIFT=0)</td>
</tr>
</tbody>
</table>

### ARM Encoding

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COND</td>
<td>Condition guard (see Section 3)</td>
</tr>
<tr>
<td>I</td>
<td>Immediate bit</td>
</tr>
<tr>
<td></td>
<td>0: SHIFT is 8-bit immediate with rotation</td>
</tr>
<tr>
<td></td>
<td>1: SHIFT is shift value</td>
</tr>
<tr>
<td>S</td>
<td>Set Condition-Codes bit</td>
</tr>
<tr>
<td></td>
<td>0: do not alter flags</td>
</tr>
<tr>
<td></td>
<td>1: set flags</td>
</tr>
</tbody>
</table>

---

Dr. Gordon Cichon  
gordon.cichon@ifi.lmu.de  
https://www.tcs.ifi.lmu.de/teaching/summer-2016/modern
Quick Reference: Assembler

2. Load/Store Operations

2.1 Intel Addressing Modes

Each instruction has two operands. Both operands are combined using the operation specified in Section 1. The destination operand is overwritten with the result (use MOV instruction to make a copy if needed!). One of the operands needs to be a register or immediate, the other one can be a load/store operation according to the following addressing modes:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>register</td>
<td>value from register, no memory access</td>
<td>SUB ECX, EDX</td>
</tr>
<tr>
<td>immediate</td>
<td>value from instruction</td>
<td>MOV EAX, 7</td>
</tr>
<tr>
<td>direct addressing</td>
<td>immediate address</td>
<td>XOR EAX, [label]</td>
</tr>
<tr>
<td>based addressing</td>
<td>register contains address</td>
<td>ADD EAX, [ESI]</td>
</tr>
<tr>
<td>base+displacement</td>
<td>register+offset address</td>
<td>MOV EAX, [EBP]8</td>
</tr>
<tr>
<td>index+displacement</td>
<td>address: base+shifted register (by 1,2,4,8)</td>
<td>SUB array[EDI], 1</td>
</tr>
<tr>
<td>base+displacement+index</td>
<td>address: base+register + shifted register (by 1,2,4,8)</td>
<td>INC array[EBX+EDI*4]</td>
</tr>
</tbody>
</table>

2.2 ARM Data Transfer Instruction

ARM is a Load/Store architecture. Consequently, each memory operand needs to be loaded into a register by a separate data transfer instruction. In recent instruction set extensions, there are also instructions for 16-bit accesses. There are also special instructions for multi-register transfers.

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COND</td>
<td>Condition guard (see Section 3)</td>
</tr>
<tr>
<td>I</td>
<td>Immediate bit</td>
</tr>
<tr>
<td></td>
<td>0: SHIFT is 12-bit immediate</td>
</tr>
<tr>
<td></td>
<td>1: SHIFT is shift value</td>
</tr>
<tr>
<td>P</td>
<td>Pre/Post Indexing bit</td>
</tr>
<tr>
<td></td>
<td>0: add offset after access (post)</td>
</tr>
<tr>
<td></td>
<td>1: add offset before access (pre)</td>
</tr>
<tr>
<td>U</td>
<td>Up/Down bit</td>
</tr>
<tr>
<td></td>
<td>0: subtract offset from base (down)</td>
</tr>
<tr>
<td></td>
<td>1: add offset to base (up)</td>
</tr>
<tr>
<td>B</td>
<td>Byte/Word bit</td>
</tr>
<tr>
<td></td>
<td>0: word access</td>
</tr>
<tr>
<td></td>
<td>1: byte access (sign extension)</td>
</tr>
</tbody>
</table>

Dr. Gordon Cichon  
gordon.cichon@ifi.lmu.de  
https://www.tcs.ifi.lmu.de/teaching/summer-2016/modern
Quick Reference: Assembler

<table>
<thead>
<tr>
<th>W</th>
<th>Write-back bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>no write-back of computed address</td>
</tr>
<tr>
<td>1</td>
<td>do write-back of computed address</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>L</th>
<th>Load/Store bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>store to memory</td>
</tr>
<tr>
<td>1</td>
<td>load from memory</td>
</tr>
</tbody>
</table>

3. Branch Instructions

3.1 Control Transfer

<table>
<thead>
<tr>
<th></th>
<th>Intel</th>
<th>ARM</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMP</td>
<td>unconditional branch</td>
<td>B</td>
</tr>
<tr>
<td>CALL</td>
<td>subroutine call</td>
<td>BL</td>
</tr>
<tr>
<td>RET</td>
<td>return from subroutine</td>
<td>(MOV PC, LR)</td>
</tr>
</tbody>
</table>

Intel has a JMP instruction with different addressing modes. ARM can move and data-processing instruction to write R15/PC register. For large offsets, and for subroutine calls, there is a specialized branch instruction.

Intel has a specialized CALL instruction that saves the return address on the stack (memory access). ARM has a “branch-and-link” instruction that saves the return address in R14/LR. The program needs to save it explicitly on the stack using a store instruction. On Intel, RET pops this saved address from the stack and continues at the caller. On ARM, the program needs to move the saved address register into the R15/PC register).

3.2 Condition Codes

On ARM, each instruction, also data-processing and load/store instructions, are guarded by a condition. This means, the guarded instruction is only executed if the condition is met. On Intel, only JMP (and MOV) operations can be guarded by condition codes.

<table>
<thead>
<tr>
<th></th>
<th>Intel</th>
<th>ARM</th>
<th>(encoding COND)</th>
</tr>
</thead>
<tbody>
<tr>
<td>JZ, JE</td>
<td>zero set, equal</td>
<td>EQ</td>
<td>0000</td>
</tr>
<tr>
<td>JNZ, JNE</td>
<td>zero clear, not equal</td>
<td>NE</td>
<td>0001</td>
</tr>
<tr>
<td>JC, JAE, JNB</td>
<td>carry set, unsigned &gt;=</td>
<td>CS</td>
<td>0010</td>
</tr>
<tr>
<td>JNC, JB, JNAE</td>
<td>carry clear, unsigned &lt;</td>
<td>CC</td>
<td>0011</td>
</tr>
<tr>
<td>JS</td>
<td>negative &lt;0</td>
<td>MI</td>
<td>0100</td>
</tr>
<tr>
<td>JNS</td>
<td>&gt;= 0</td>
<td>PL</td>
<td>0101</td>
</tr>
<tr>
<td>JO</td>
<td>overflow set</td>
<td>VS</td>
<td>0110</td>
</tr>
<tr>
<td>JNO</td>
<td>overflow clear</td>
<td>VC</td>
<td>0111</td>
</tr>
</tbody>
</table>

Dr. Gordon Cichon

gordon.cichon@ifi.lmu.de

https://www.tcs.ifi.lmu.de/teaching/summer-2016/modern
## Quick Reference: Assembler

<table>
<thead>
<tr>
<th>Condition</th>
<th>Comparison</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>JA, JNBE</td>
<td>unsigned &gt;</td>
<td>HI 1000</td>
</tr>
<tr>
<td>JBE</td>
<td>unsigned &lt;=</td>
<td>LS 1001</td>
</tr>
<tr>
<td>JGE, JNL</td>
<td>signed &gt;=</td>
<td>GE 1010</td>
</tr>
<tr>
<td>JL, JNGE</td>
<td>signed &lt;</td>
<td>LT 1011</td>
</tr>
<tr>
<td>JG, JNLE</td>
<td>signed &gt;</td>
<td>GT 1100</td>
</tr>
<tr>
<td>JLE, JNG</td>
<td>signed &lt;=</td>
<td>LE 1101</td>
</tr>
<tr>
<td></td>
<td>always</td>
<td>(AL) 1110</td>
</tr>
<tr>
<td></td>
<td>never (reserved)</td>
<td>1111</td>
</tr>
</tbody>
</table>