Assembly Language
Assembly Language and Assembler

- **Assembly language**
  - Documents machine instructions using agreed-up-on syntax
  - Translation from an assembly instruction to a machine instruction is straightforward
    - One-to-one correspondence
    - Assembler (a computer program) does the translation
    - A translator that translates the program in an assembly language into the program in machine language
SNUVM#1 ISA - Registers

- Registers
  - 17 32-bit registers
    - r0 - r15
      - To store data or addresses
    - cpsr
      - processor status

- Special purpose registers
  - 4 32-bit registers
    - r13 (stack pointer)
    - r14 (link register)
    - r15 (program counter)
SNUVM#1 ISA - Registers (contd.)

- pc - program counter
  - Instruction pointer
  - The address of current instruction + 8

- cpsr - current processor status register
  - Contains N, Z, C, and V bits (CPSR[31:28])
    - N (negative) - the result of the last ALU operation is negative (MSB = 1)
    - Z (zero) - the result of the last ALU operation is zero
    - C (carry) - the result of the last ALU operation or shift operation has carry-out
    - V (overflow) - the result of the last ALU operation overflows

\[
\begin{array}{cccc}
31 & 30 & 29 & 28 \\
N & Z & C & V \\
\end{array}
\]

...
SNUVM#1 ISA - Memory

- Memory
  - Does not distinguish instructions from data
    - Unified instruction and data memory
  - Each byte has its own address
    - Byte addressable
  - Byte ordering
    - Little Endian
  - 32-bit Addresses
    - 0x00000000 - 0x000FFFFF (1MB)
  - The first instruction in the program is loaded at 0x8000
Byte Ordering

- **Big Endian**
  - Least significant byte has highest address
  - Sun’s, Mac’s are “Big Endian” machines
- **Little Endian**
  - Least significant byte has lowest address
  - Alphas, PC’s are “Little Endian” machines
- A 32-bit word 0x12345678

<table>
<thead>
<tr>
<th></th>
<th>0x100</th>
<th>0x101</th>
<th>0x102</th>
<th>0x103</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Big Endian</strong></td>
<td>12</td>
<td>34</td>
<td>56</td>
<td>78</td>
</tr>
<tr>
<td><strong>Little Endian</strong></td>
<td>78</td>
<td>65</td>
<td>43</td>
<td>21</td>
</tr>
</tbody>
</table>
SNUVM#1 ISA - Shifter

ALU

Shifter

shifter operand
Some Notations

- `<imm_n>`
  - An n-bit constant
- `<Rm>`
  - Register m
Shifter Operands <shifter_operand>

- #<imm_8>
- <Rm>
- <Rm>, LSL #<imm_5>
  - logical shift left by <imm_5>
  - R5, LSL #5
- <Rm>, LSL <Rs>
  - logical shift left by the value Rs[4:0]
  - R5, LSL R2
- <Rm>, LSR #<imm_5>
  - logical shift right by <imm_5>
  - R5, LSR #4
Shifter Operands (contd.)

- `<Rm>, LSR <Rs>`
  - logical shift right by the value Rs[4:0]
  - R5, LSR R3
- `<Rm>, ASR #<imm_5>`
  - arithmetic shift right by `<imm_5>`
  - R5, ASR #0x3
- `<Rm>, ASR <Rs>`
  - arithmetic shift right by the value Rs[4:0]
  - R5, ASR R0
SNUVM#1 ISA - Instructions

- 32-bit length
- Data processing instructions
  - Move instructions
  - Arithmetic instructions
  - Logical instructions
  - Comparison instructions
- Branch instructions
- Load and store instructions
Move

- **MOVS <Rd>, <shifter_operand>**
  - Moves the value of <shifter_operand> to the destination register <Rd>
    - <Rd>: R0, ..., R14
  - The condition flags are updated based on the result
    - N, Z
    - C - the shifter carry out
    - V - not affected
- **Examples**
  - MOVS r6, #0x24
  - MOVS r7, #12
  - MOVS r6, r4
  - MOVS r7, r5, LSL #2
Move (contd.)

MOVS r7, r5, LSL #2

```
MOVS r7, r5, LSL #2

00000000000000000000000000000110
```

```
r5, LSL #2
00000000000000000000000000000110
```

```
r5 0x00000006
r7 0x00000009
cpsr xxxxXXXXXXXX
```

```
r5 0x00000006
r7 0x00000018
cpsr nzcxXXXXXXXX
```

```
00000000000000000000000000000110
```

```
r5, LSL #2
00000000000000000000000000000110
```

```
r5 0x00000006
r7 0x00000018
cpsr nzcxXXXXXXXX
```
Arithmetic Instructions

- **ADDS <Rd>, <Rn>, <shifter_operand>**
  - Adds the value of <shifter_operand> to the value of register <Rn>, and stores the result in the destination register <Rd>
  - <Rd>: R0, ..., R14
  - The condition flags are updated based on the result
    - N, Z
    - C - the carry out from Rn + shifter_operand
    - V - overflow from Rn + shifter_operand

- **Examples**
  - ADDS r6, r7, #0x24
  - ADDS r7, r8, #12
  - ADDS r6, r4, r3
  - ADDS r7, r0, r5, LSL #2
**Arithmetic Instructions (contd.)**

**ADDS r0, r1, r2, LSL #1**

```
  r0   0x00000000
  r1   0x00000003
  r2   0x00000001
  ...  
  cpsr xxxxXXXXXXXX
```

```
  r0   0x00000005
  r1   0x00000003
  r2   0x00000001
  ...  
  cpsr nzcvXXXXXXXX
```
Arithmetic Instructions (contd.)

- **SUBS <Rd>, <Rn>, <shifter_operand>**
  - Subtracts the value of `<shifter_operand>` from the value of register `<Rn>`, and stores the result in the destination register `<Rd>`
    - `<Rd>`: R0, ..., R14
  - The condition flags are updated based on the result
    - Similar to ADDS
Arithmetic Instructions (contd.)

SUBS r0, r1, #1

<table>
<thead>
<tr>
<th>r0</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>0x00000001</td>
</tr>
<tr>
<td>cpsr</td>
<td>xxxXxxxxxxxx</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>r0</th>
<th>0x00000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>0x00000001</td>
</tr>
<tr>
<td>cpsr</td>
<td>nZCvXXXXXXXX</td>
</tr>
</tbody>
</table>

Computer Principles
Logical Instructions

- **ANDS <Rd>, <Rn>, <shifter Operand>**
  - Performs bitwise AND of the value of <shifter_operand> with the value of register <Rn>, and stores the result in the destination register <Rd>
  - <Rd>: R0, ..., R14
  - The condition flags are updated based on the result
    - N, Z
    - C - the shifter carry out
    - V - not affected
Logical Instructions (contd.)

- **ORRS** `<Rd>, <Rn>, <shifter_operand>`
  - Performs bitwise OR of the value of `<shifter_operand>` with the value of register `<Rn>`, and stores the result in the destination register `<Rd>`
    - `<Rd>`: R0, ..., R14
  - The condition flags are updated based on the result
    - The same way as in **ANDS**

- **EORS** `<Rd>, <Rn>, <shifter_operand>`
  - Performs bitwise XOR of the value of `<shifter_operand>` with the value of register `<Rn>`, and stores the result in the destination register `<Rd>`
    - `<Rd>`: R0, ..., R14
  - The condition flags are updated based on the result
    - The same way as in **ANDS**
Logical Instructions (contd.)

ORRS r0, r1, r2

<table>
<thead>
<tr>
<th>r0</th>
<th>...</th>
<th>r0</th>
<th>0x22446688</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>0x02040608</td>
<td>r1</td>
<td>0x02040608</td>
</tr>
<tr>
<td>r2</td>
<td>0x20406080</td>
<td>r2</td>
<td>0x20406080</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>cpsr</td>
<td>xxxxXXXXXXXX</td>
<td>cpsr</td>
<td>nzcxXXXXXXXX</td>
</tr>
</tbody>
</table>

Computer Principles
Comparison Instruction

- **CMP <Rn>, <shifter_operand>**
  - Compares the value of `<shifter_operand>` with the value of register `<Rn>`, and updates the condition flags based on the result of `<Rn> - <shifter_operand>`
    - `<Rn>`: R0, ..., R15
  - The condition flags are updated based on the result
    - Similar to SUBS
Comparison Instruction (contd.)

CMP r1, r4

...  ...  ...
  r1  0x00000005  r1  0x00000005  ...
  ...  ...  ...
  r4  0x00000005  r4  0x00000005  ...
  ...  ...  ...
  cpsr  xxxxxxxxxxxx  cpsr  nZCvxxxxxxxx
Branch Instructions

- B <label>
  - <label> is a name
  - Converted by the assembler to the target address
  - Forward and backward branches up to 32MB
  - Causes a jump to the target address
  - No effects on the condition flags
Branch Instructions (contd.)

- **B<cond> <label>**
  - `<cond>` is the condition under which the instruction is executed
  - Causes a branch to the target address if the condition matches the condition flags of the cpsr
  - No effects on the condition flags

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Name</th>
<th>Condition flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQ</td>
<td>Equal</td>
<td>Z</td>
</tr>
<tr>
<td>NE</td>
<td>Not equal</td>
<td>z</td>
</tr>
<tr>
<td>GE</td>
<td>Signed greater than or equal</td>
<td>NV or nv</td>
</tr>
<tr>
<td>LT</td>
<td>Signed less than</td>
<td>Nv or nV</td>
</tr>
<tr>
<td>GT</td>
<td>Signed greater than</td>
<td>NzV or nzv</td>
</tr>
<tr>
<td>LE</td>
<td>Signed less than or equal</td>
<td>Z or Nv or nV</td>
</tr>
</tbody>
</table>
Branch Instructions (contd.)

```
ADD S r2, r3, r0
B L1
ADD S r1, r2, #4
...
L1:
SUB S r1, r2, #4
```
Programming Example

- Summing the numbers ranging from 1 to 100

```
MOVS R0, 100
MOVS R1, #0
MOVS R2, #0

LOOP:
  ADDS R2, R2, #1
  ADDS R1, R1, R2
  CMP  R0, R2
  BNE  LOOP
```
Load Instructions

- **LDR <Rd>, [ <Rn> ]**
  - Makes 32-bit memory data located at the address contained in <Rn> to be loaded into the destination register <Rd>
  - <Rd>: R0 - R15

- **LDRH <Rd>, [ <Rn> ]**
  - Loads a halfword (16 bits) from memory and zero-extends it to form a 32-bit word, and then stores the result to the destination register <Rd>

- **LDRB <Rd>, [ <Rn> ]**
  - Loads a byte from memory and zero-extends it to form a 32-bit word, and then stores the result to the destination register <Rd>
Load Instructions (contd.)

- **LDRSH <Rd>, [ <Rn> ]**
  - Loads a halfword (16 bits) from memory and sign-extends it to form a 32-bit word, and then stores the result to the destination register <Rd>

- **LDRSB <Rd>, [ <Rn> ]**
  - Loads a byte from memory and sign-extends it to form a 32-bit word, and then stores the result to the destination register <Rd>
Load Instructions (contd.)

LDR r0, [r1]

<table>
<thead>
<tr>
<th>r0</th>
<th>...</th>
<th>r0</th>
<th>0x00000004</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>0x00009000</td>
<td>r1</td>
<td>0x00009000</td>
</tr>
</tbody>
</table>

M32[0x9000] 0x00000004

M32[0x9000] 0x00000004
Store Instructions

- **STR <Rd>, [ <Rn> ]**
  - Makes 32-bit data from the register <Rd> to be stored to the memory location with the address contained in <Rn>
  - <Rd>: R0, ..., R15

- **STRH <Rd>, [ <Rn> ]**
  - Makes 16-bit data from the register <Rd> (Rd[15:0]) to be stored to the memory location with the address contained in <Rn>

- **STRB <Rd>, [ <Rn> ]**
  - Makes 8-bit data from the register <Rd> (Rd[7:0]) to be stored to the memory location with the address contained in <Rn>