N-105

Rosemary Francis
Computer Laboratory
University of Cambridge
August 2004
N-105

1 Introduction

The N-105 is a 16-bit embedded soft core processor with a two stage pipeline and a RISC instruction set. It was designed by Rosemary Francis, Daniel Hulme and Simon Moore as a teaching resource. Its is loosely based on Altera’s Nios, but contains no derived code. It is designed to be used with Altera’s Avalon bus as it has no memory management. It assumes logically separate data and instruction memories and has two independent interfaces to the bus. The two stage pipeline consists of instruction fetch and everything else.

2 Overview

2.1 Registers

There are 16 16-bit general purpose registers and one reserved register called the pc (program counter). The general purpose registers are named r0 to r15. R15 may be read or written to in the same way others are. During a branch to subroutine(BSR) the pc+2 is written to r15 and then r15 is written to pc on a return (RET). You can therefore manually write to the pc using the RET instruction. It is up to the programmer to ensure the contents of r15 is valid during a return. The pc contains the address of the next instruction to execute. It is half word aligned.

2.2 Condition Flags

The condition flags are set by selected alu operations. Some set all flags, others set only a few flags. In the case where only a few are set, the value of those not set will be undefined. If an instruction does not set any flags then they are unaffected and retain the value at which they were last set.

<table>
<thead>
<tr>
<th>Condition Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
</tr>
<tr>
<td>V</td>
</tr>
<tr>
<td>Z</td>
</tr>
<tr>
<td>C</td>
</tr>
</tbody>
</table>
2.3 Condition codes

There is only one condition instruction: IFS. This allows the instruction following to be executed only if the condition is true. A skipped instruction is still fetched from memory and so there is no time advantage unless the skipped instruction was a load or store.

<table>
<thead>
<tr>
<th>Condition Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>code</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>0000</td>
</tr>
<tr>
<td>0001</td>
</tr>
<tr>
<td>0010</td>
</tr>
<tr>
<td>0011</td>
</tr>
<tr>
<td>0100</td>
</tr>
<tr>
<td>0101</td>
</tr>
<tr>
<td>0110</td>
</tr>
<tr>
<td>0111</td>
</tr>
<tr>
<td>0110</td>
</tr>
<tr>
<td>0111</td>
</tr>
<tr>
<td>0100</td>
</tr>
<tr>
<td>0101</td>
</tr>
<tr>
<td>0100</td>
</tr>
<tr>
<td>0101</td>
</tr>
</tbody>
</table>

2.4 Program Flow

There are three branch instructions: BR, BSR and RET. These all update the pc. However the pc contains the value of the the next instruction to execute which is the instruction currently being fetched. This means the the instruction following a branch will be executed as the pipeline is not flushed. Returns therefore return to the instruction after so that not instructions are executed twice. For example in the code below 8 will be moved to r4 before the pc will point to instruction at three. Also 4 will be moved to r1 before the return takes place. The return sets the pc to the instruction at two. Here nop is a “no op” which is equivalent to “or r0 r0” which does nothing.

```
two movi r4 6
   movi r9 7
   br end
   nop
three ret
   movi r1 4
end     nop
```
2.5 Memory Access

Memory access is via Altera’s Avalon bus used for their NIOS soft processor. The Avalon bus can handle 16 bit transfers only so addresses must be word aligned. Apart from that there is no memory management. Transfers may only take place when the bus is ready the processor may have to wait indefinitely. Loads and stores typically take only a few clock cycles longer. Instruction fetch works similarly. It typically takes 2 clock cycles to fetch an instruction.

2.6 Instruction Fields

A Index of source and destination register
B Index of source register
imm4 4-bit unsigned immediate
imm7 7-bit signed or unsigned immediate
imm11 11-bit signed immediate

Instruction formats

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>imm4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>imm7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>imm11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

opcode

opcode

opcode

opcode

opcode
## 3 Instruction Set Quick Reference

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Mnemonic</th>
<th>Operands</th>
<th>Function</th>
<th>Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>000 00</td>
<td>OR</td>
<td>ra rb</td>
<td>ra ← ra</td>
<td>n-z-</td>
</tr>
<tr>
<td>000 01</td>
<td>AND</td>
<td>ra rb</td>
<td>ra ← ra &amp; rb</td>
<td>n-z-</td>
</tr>
<tr>
<td>000 10</td>
<td>XOR</td>
<td>ra rb</td>
<td>ra ← ra ⊕ rb</td>
<td>n-z-</td>
</tr>
<tr>
<td>000 11</td>
<td>NOT</td>
<td>ra</td>
<td>ra ← !ra</td>
<td></td>
</tr>
<tr>
<td>001 00</td>
<td>MOV</td>
<td>ra rb</td>
<td>ra ← rb</td>
<td></td>
</tr>
<tr>
<td>001 01</td>
<td>ADD</td>
<td>ra rb</td>
<td>ra ← ra + rb</td>
<td>nvzc</td>
</tr>
<tr>
<td>001 10</td>
<td>SUB</td>
<td>ra rb</td>
<td>ra ← ra − rb</td>
<td>nvzc</td>
</tr>
<tr>
<td>001 11</td>
<td>CMP</td>
<td>ra rb</td>
<td>ra − rb</td>
<td>nvzc</td>
</tr>
<tr>
<td>010 00</td>
<td>LSLI</td>
<td>ra imm4</td>
<td>ra ← ra &lt;&lt; 16-imm4</td>
<td></td>
</tr>
<tr>
<td>010 01</td>
<td>LSRI</td>
<td>ra imm4</td>
<td>ra ← ra &gt;&gt; imm4</td>
<td></td>
</tr>
<tr>
<td>010 10</td>
<td>ASRI</td>
<td>ra imm4</td>
<td>ra ← ra &gt;&gt;&gt; imm4</td>
<td></td>
</tr>
<tr>
<td>010 11</td>
<td>ROTI</td>
<td>ra imm4</td>
<td>ra ← ra rotated &gt; imm4</td>
<td></td>
</tr>
<tr>
<td>011 00</td>
<td>MOVI</td>
<td>ra imm7</td>
<td>ra ← imm7</td>
<td></td>
</tr>
<tr>
<td>011 01</td>
<td>ADDI</td>
<td>ra imm7</td>
<td>ra ← ra + imm7</td>
<td>nvzc</td>
</tr>
<tr>
<td>011 10</td>
<td>SUBI</td>
<td>ra imm7</td>
<td>ra ← ra − imm7</td>
<td>nvzc</td>
</tr>
<tr>
<td>011 11</td>
<td>CMPI</td>
<td>ra imm7</td>
<td>ra − imm7</td>
<td>nvzc</td>
</tr>
<tr>
<td>100 00</td>
<td>LD</td>
<td>ra rb</td>
<td>ra ← [rb]</td>
<td></td>
</tr>
<tr>
<td>101 00</td>
<td>ST</td>
<td>ra rb</td>
<td>[rb] ← ra</td>
<td></td>
</tr>
<tr>
<td>110 00</td>
<td>BR</td>
<td>imm11</td>
<td>pc ← pc + imm11</td>
<td></td>
</tr>
<tr>
<td>110 01</td>
<td>BSR</td>
<td>imm11</td>
<td>pc ← pc + imm11</td>
<td></td>
</tr>
<tr>
<td>110 10</td>
<td>RET</td>
<td>imm11</td>
<td>r15 ← pc + 2</td>
<td></td>
</tr>
<tr>
<td>111 11</td>
<td>IFS</td>
<td>cc_imm4</td>
<td>skip ← !condition</td>
<td></td>
</tr>
</tbody>
</table>
ADD

Operation: \( rA \leftarrow rA + rB \)

Assembler Syntax: `add ra rb`

Example: `add r4 r5`

Description: Add the values contained in rA and rB and place the result in rA

Condition codes: Flags set

Instruction Fields: Register index A, register index B

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>x</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ADDI

ADD immediate

Operation: \[ rA \leftarrow rA + \text{imm7} \]

Assembler Syntax: `addi rA imm7`

Example: `addi r8 3`

Description: Add the values contained in \( rA \) and \( \text{imm7} \) and place the result in \( rA \)

Condition codes: Flags set

Instruction Fields: register index A, unsigned 7-bit immediate \( \text{imm7} \)

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>imm7</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
AND

Operation: \( rA \leftarrow rA \& rB \)

Assembler Syntax: `and rA rB`

Example: `and r14 r0`

Description: AND the bits in \( rA \) and \( rB \) and place the result in \( rA \)

Condition codes: Negative and zero flags set, others set undefined

Instruction Fields: register index A, register index B

| \( A \) | \( B \) | \( x \) | \( 0 \) | \( 0 \) | \( 0 \) | \( 0 \) | \( 1 \) |
ASRI

Arithmetic right shift

Operation: \( rA \leftarrow rA >> \text{imm4} \)

Assembler Syntax: \text{asri } rA \text{ imm4}

Example: \text{asri } r6 \text{ 9}

Description: shift \( rA \) right by the amount in \( \text{imm4} \), shifting in the 15th bit to preserve the sign

Condition codes: Flags unaffected

Instruction Fields: Register index A, 4-bit unsigned immediate \( \text{imm4} \)

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>imm4</td>
<td>x</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
BR

Branch

Operation: \( \text{pc} \leftarrow \text{pc} + \text{imm11} \)

Assembler Syntax: \( \text{br} \ addr \)

Example: \( \text{br} \ \text{loopLabel} \)

Description: Performs a branch to addr. The delay slot is then executed

Condition codes: Flags unaffected

Instruction Fields: Signed 11-bit immediate \( \text{imm11} \)

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\( \text{imm11} \) 11 00 00
BSR

Branch to subroutine

Operation: \( \text{pc} \leftarrow \text{pc} + \text{imm11} \)  
\( \text{r15} \leftarrow \text{pc} + 2 \)

Assembler Syntax: \text{bsr label}

Example: \text{bsr loopLabel}

Description: Branch to addr and store pc in r15. The delay slot is then executed

Condition codes: Flags unaffected

Instruction Fields: Signed 11 bit immediate imm11

\[
\begin{array}{cccccccccccc}
15 & 14 & 13 & 12 & 11 & 10 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\
\hline
\text{imm11} & 1 & 1 & 0 & 0 & 1
\end{array}
\]
**CMP**

**Operation:** \( rA - rB \)

**Assembler Syntax:** `cmp rA rB`

**Example:** `cmp r9 r3`

**Description:** Subtract \( rB \) from \( rA \) but do not store the result.

**Condition codes:** Flags set

**Instruction Fields:** Register index A, register index B

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>x</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Compare**
CMPI

Compare immediate

Operation: \( rA - \text{imm7} \)

Assembler Syntax: `cmpi rA imm7`

Example: `cmpi r9 -24`

Description: Subtract imm7 from the contents of rA, but do not store the result

Condition codes: Flags set

Instruction Fields: Register index A, signed 7-bit immediate imm7

\[
\begin{array}{cccccccccccccc}
15 & 14 & 13 & 12 & 11 & 10 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\
A & \text{imm7} & 0 & 1 & 1 & 1 & 1 & 1
\end{array}
\]
IFS

Conditionally execute next instruction

**Operation:** skip ← !condition

**Assembler Syntax:** `ifs cc_imm4`

**Example:** `ifs cc_gt`

**Description:** Skip the next instruction if the condition is false, execute if true

**Condition codes:** Flags unaffected

**Instruction Fields:** 4-bit condition immediate imm4

```
15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
  x  | imm4  |  x  | 1 | 1 | 1 | 1 | 1
```
**LD**

**Load**

**Operation:** \( rA \leftarrow [rB] \)

**Assembler Syntax:** `ld rA rB`

**Example:** `ld r5 r6`

**Description:** Load the half word at address in \( rB \) into \( rA \). Address must be half word aligned.

**Condition codes:** Flags unaffected

**Instruction Fields:** Register index \( A \), register index \( B \)

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**LSLI**

**Logical left shift**

**Operation:** \( rA \leftarrow rA \ll 16 - \text{imm4} \)

**Assembler Syntax:** `lsli rA 16, imm4`

**Example:** `lsli r11, 13`

**Description:** Logical left shift by a 4-bit immediate (shifts in 0s)
The four bit immediate is the bottom four bits of 16-imm4

**Condition codes:** Flags unaffected

**Instruction Fields:** Register index A, 4-bit immediate

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
LSRI

Logical right shift

Operation: \( rA \leftarrow rA \gg \text{imm4} \)

Assembler Syntax: `lsri rA imm4`

Example: `lsri r0 5`

Description: Logical shift right by a 4-bit immediate imm4 (shifts in 0s)

Condition codes: Flags unaffected

Instruction Fields: Register index rA, 4-bit immediate imm4

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>imm4</td>
<td>x</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
MOV

Operation: rA ← rB

Assembler Syntax: mov rA rB

Example: mov r8 r2

Description: Move the contents of rB into rA

Condition codes: Flags unaffected

Instruction Fields: Register index rA, register index rB

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>x</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
MOVI

Move immediate

Operation: \( rA \leftarrow \text{imm7} \)

Assembler Syntax: `movi rA imm7`

Example: `movi r12 41`

Description: Move 7-bit signed immediate \( \text{imm7} \) into \( rA \) (with sign extension)

Condition codes: Flags unaffected

Instruction Fields: Register index A, signed 7-bit immediate \( \text{imm7} \)

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>imm7</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**NOP**

**Operation:** none

**Assembler Syntax:** nop

**Example:** nop

**Description:** Does nothing. Equivalent to “or r0 r0”

**Condition codes:** Negative and zero flags set, others left undefined

**Instruction Fields:** none

```
  15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```
**NOT**

**Bitwise Logical NOT**

**Operation:** \( rA \leftarrow \neg rA \)

**Assembler Syntax:** `not rA`

**Example:** `not r3`

**Description:** Logical not, bitwise inversion

**Condition codes:** Flags unaffected

**Instruction Fields:** Register index A

<table>
<thead>
<tr>
<th></th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

21
OR

Bitwise Logical OR

Operation: \( rA \leftarrow rA \mid rB \)

Assembler Syntax: `or rA rB`

Example: `or r9 r2`

Description: Perform bitwise logical or on bits in rA with bits in rB and place in rA.

Condition codes: Negative and zero flags set, others left undefined.

Instruction Fields: Register index A, register index B

<table>
<thead>
<tr>
<th></th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
RET

Return from Subroutine

Operation: pc ← r15

Assembler Syntax: ret

Example: ret

Description: Returns from subroutine by writing address saved in r15 to the program counter

Condition codes: Flags unaffected

Instruction Fields: none

```
15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
  x | 1 | 1 | 0 | 1 | 1
```
### ROTI

**Bitwise Rotate Right**

**Operation:**

\[ rA \leftarrow rA >> \text{imm4} \mid ra <<= 16-\text{imm4} \]

**Assembler Syntax:**

`roti rA imm4`

**Example:**

`roti r9 11`

**Description:**

Rotate the bits in `rA` right `imm4` places

**Condition codes:**

Flags unaffected

**Instruction Fields:**

Register index A, 4-bit unsigned immediate `imm4`

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>imm4</td>
<td>x</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


**ST**

**Operation:** \[ rB ] ← rA

**Assembler Syntax:** st rA rB

**Example:** st r15 r8

**Description:** Store the value in rA at address in rB. Address must be half word aligned

**Condition codes:** Flags unaffected

**Instruction Fields:** Register index A, register index B

```
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

A B x 1 0 1 0 0
```
SUB

Subtract

Operation: \( rA \leftarrow rA - rB \)

Assembler Syntax: `sub rA rB`

Example: `sub r2 r0`

Description: Subtract \( rB \) from \( rA \) and store the result back in \( rA \)

Condition codes: Flags set

Instruction Fields: Register index A, register index B

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>x</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

26
SUBI

Subtract immediate

Operation: \( rA \leftarrow rA - \text{imm7} \)

Assembler Syntax: \( \text{subi } rA \text{ imm7} \)

Example: \( \text{subi } r9 \ 63 \)

Description: Subtracts unsigned 7-bit immediate \( \text{imm7} \) from \( rA \) and stores result back in \( rA \)

Condition codes: Flags set

Instruction Fields: Register index \( rA \), unsigned 7-bit immediate \( \text{imm7} \)

\[
\begin{array}{cccccccccccc}
15 & 14 & 13 & 12 & 11 & 10 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\
A & \text{imm7} & 0 & 1 & 1 & 1 & 0
\end{array}
\]
XOR

Bitwise Logical Exclusive OR

Operation: \( rA \leftarrow rA \oplus rB \)

Assembler Syntax: `xor rA rB`

Example: `xor r0 r6`

Description: Performs logical bitwise exclusive or on \( rA \) and \( rB \) and stores the result back in \( rA \)

Condition codes: Negative and zero flags set, others left undefined

Instruction Fields: Register index A, register index B