

# Project Overview for ECE 152

## The Architecture of the Duke152/16 (revised Dec 13, 2007)

The project for this course is the design and implementation of the Duke152/16, a 16-bit MIPS-like architecture that has been scaled down and simplified to make it feasible for a class project. I have specified the architecture, and you will incrementally build a computer that implements this architecture. The architecture's instructions are in Table 1.

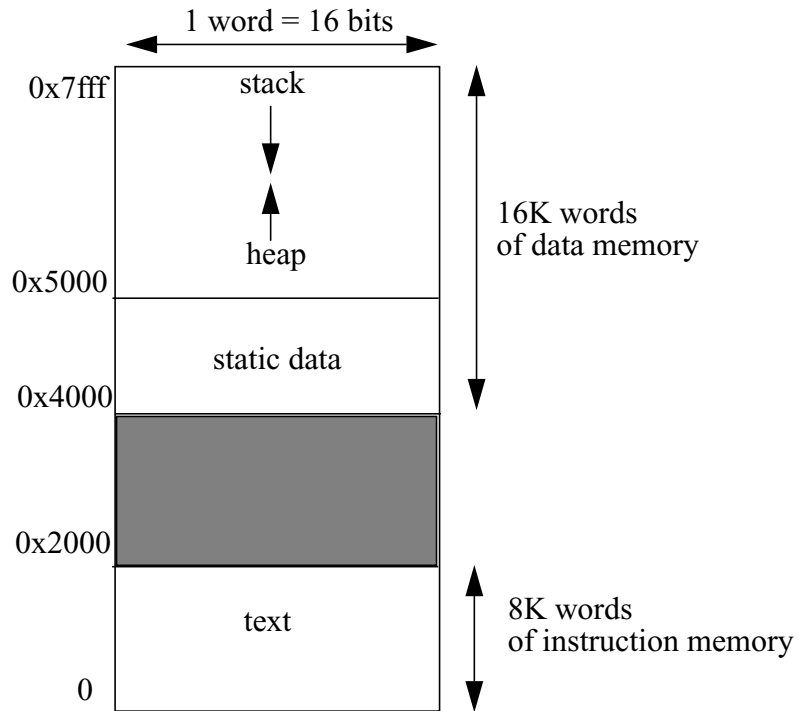
The formats of the R, I, and J type instructions are shown in Figure 1. The branch instruction, `bnez`, is NOT a “delayed branch” —that is, the instruction immediately following a branch is NOT executed if the branch is taken.

**TABLE 1. Duke152/16 Instructions**

instruction	opcode	type	usage	operation
<code>add</code>	0001	R	<code>add \$rd, \$rs, \$rt</code>	$\$rd = \$rs + \$rt$
<code>ldi</code>	0010	I	<code>ldi \$rt, 1</code>	$\$rt = 1$
<code>sub</code>	0011	R	<code>sub \$rd, \$rs, \$rt</code>	$\$rd = \$rs - \$rt$
<code>and</code>	0100	R	<code>and \$rd, \$rs, \$rt</code>	$\$rd = \$rs \text{ AND } \$rt$
<code>xor</code>	0101	R	<code>xor \$rd, \$rs, \$rt</code>	$\$rd = \$rs \text{ XOR } \$rt$
<code>rotl</code>	0110	R	<code>rotl \$rd, \$rs, 2</code>	$\$rd = \$rs \text{ rotated } 2 \text{ to left}$
<code>rotr</code>	0111	R	<code>rotr \$rd, \$rs, 3</code>	$\$rd = \$rs \text{ rotated } 3 \text{ to right}$
<code>lw</code>	1000	I	<code>lw \$rt, 10(\$rs)</code>	$\$rt = \text{Mem}[\$rs+10]$
<code>sw</code>	1001	I	<code>sw \$rt, 10(\$rs)</code>	$\text{Mem}[\$rs+10] = \$rt$
<code>bnez</code>	1010	I	<code>bnez \$rs, b</code>	if $(\$rs \neq 0)$ then $\text{PC} = \text{PC} + 1 + b$
<code>j</code>	1011	J	<code>j L</code>	$\text{PC} = L$ (upper 4 bits same)
<code>ret</code>	1100	R	<code>ret</code>	$\text{PC} = \$r7$
<code>jal</code>	1101	J	<code>jal L</code>	$\$r7 = \text{PC} + 1$ ; $\text{PC} = L$
<code>input</code>	1110	I	<code>input \$rt</code>	$\$rt = \text{keyboard input}^a$
<code>output</code>	1111	I	<code>output \$rs</code>	print $\$rs$ on LCD output <sup>b</sup>

a. The read instruction is nonblocking. After a read,  $\$rt[15..9]$  will always be zero.  $\$rt[8]$  will be 0 if and only if valid data was read from the keyboard controller.  $\$rt[7..0]$  will contain the ASCII representation of the key read from the buffer. Therefore, if  $\$rt < 256$ ,  $\$rt$  is equal to the ASCII code (since  $\$rt[15..8]$  will be zero). On the cycle that data is read from the keyboard controller, the acknowledge line on the keyboard controller should be pulled high for exactly one cycle (to tell the FIFO to shift). It is okay to assert acknowledge even if there was no valid data in the FIFO.

b. Print the character that corresponds to the ASCII code stored in  $\$rt$  and advance the cursor on the LCD, going to the next line if necessary. If  $\$rs$  equals `0x0D` (carriage return), the effect will be that of a carriage return followed by a line feed. On the cycle that data is provided to the LCD controller, the `write_en` line on the LCD controller should be pulled high for exactly one cycle.



**FIGURE 2. Memory allocation convention**

There are 8 general purpose registers: \$r0-\$r7. The register \$r0 is always zero. The register \$r7 is the link register for the jal instruction. **The register \$r6 is always the base address of the data segment (0x4000).** While it might appear that you need another register to be always set to the top of the stack, this is not true (hint: that address can live in the static data segment).

The conventions for memory allocation, as performed by the assembler I will provide you, are shown in Figure 2. Note that you have an address space of 24 Kwords (each word is 16 bits), divided into 8K words of instruction memory and 16K words of data memory. Addresses between 0 and 0x1fff correspond to the 8Kword instruction memory, and

<b>R-type</b>	<b>opcode (4)</b>	<b>rs (3)</b>	<b>rt (3)</b>	<b>rd (3)</b>	<b>shiftamount (3)</b>
<b>I-type</b>	<b>opcode (4)</b>	<b>rs (3)</b>	<b>rt (3)</b>	<b>immediate (6)</b>	
<b>J-type</b>	<b>opcode (4)</b>	<b>address (12)</b>			

**FIGURE 1. Instruction Formats**

addresses between 0x4000 and 0x7fff correspond to the 16K word data memory (i.e., 0x4000 is the address of the first location in the data memory). Addresses between 0x2000 and 0x3fff do not correspond to locations in physical memory. For those of you wishing to write self-modifying code, you're out of luck.