RAG SOFTWARE

AEMS MACRO ASSEMBLER

9900 CPU REFERENCE

Macro Assembler
Version 1
R. A. Green

CONTENTS

January 1993

THE ADDRESS SPACE

The TMS 9900 Microprocessor is a 16-bit microprocessor, with a rich instruction set. The memory of the 9900 is organized into 16-bit words each of which contains two 8-bit bytes. The microprocessor has instructions to operate on both words and bytes. The memory addresses used by the 9900 are also 16-bit, limiting the "address space" to 65536 bytes, numbered 0 to 65535.

NUMBER SYSTEMS

The 9900, like most computers, operates on binary numbers. It can process either 8-bit (8 binary digits) bytes or 16-bit words. Binary numbers are base (or radix) two. There are only two digits, zero and one. The following table shows the binary representation of some decimal numbers.

DECIMAL	BINARY
1	1
2	10
10	1010
256	100000000

The table shows that writing numbers in binary soon gets to be a tiresome job, so that binary notation is almost never used. Decimal notation is usually used, but this requires that some program, like an Assembler or Compiler translate the decimal numbers into the computer's binary numbers.

When programming the computer, especially in Assembler Language, one often must refer to memory addresses and to individual bits in bytes or words. The table below shows some such references. Because the underlying number base is two, these common quantities are difficult numbers in the decimal system and seem unrelated to each other.

DECIMAL	QUANTITY
1024	1K
65535	Highest Address
128	First bit of a byte
64	Second bit of a byte
32768	First bit of a word
16384	Second bit of a word

To solve the problem of these difficult numbers, the hexadecimal number system is usually used when referring to numbers which represent addresses and values of individual bits, and decimal numbers are used only when dealing with the world external to the computer (i.e. to the user).

The hexadecimal system is base 16 which is a power of two. A hexadecimal digit thus exactly represents 4 binary digits or bits. A byte then is nicely represented as two hexadecimal digits and a word as four. The hexadecimal number system has 16 digits as shown below.

HEX	BINARY	DECIMAL
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

Although at first hex numbers may seem hard to do arithmetic on, with practice (and a hex calculator) it can be mastered. Some numbers that programmers must handle are easier to work with in hex than in decimal. Below we redo one of the previous tables to demonstrate this.

DECIMAL	HEX	QUANTITY
1024	>400	1K
65535	>FFFF	Highest Address
128	>80	First bit of a byte
64	>40	Second bit of a byte
32768	>8000	First bit of a word
16384	>4000	Second bit of a word

Only positive numbers have been dealt with so far, with a byte representing numbers in the range 0 to 255 (>0 to >FF) and words representing numbers in the range 0 to 65535 (>0 to >FFFF). The computer can also process negative numbers. The computer actually operates on two types of numbers with exactly the same instructions:

1. Unsigned or logical numbers in the range 0 to 255 for bytes and the range of 0 to 65535 for words.
2. Signed or two's complement numbers in the range -128 to +127 for bytes and -32768 to +32767 for words. Note that the negative values have a magnitude one greater than the positive values.

The "two's complement" representation of signed numbers is used, rather than having a sign and a value (as is ordinarily done in decimal) to simplify the computer hardware and to enhance the speed of operation. It is because two's complement notation is used that a single instruction can operate on either signed or unsigned numbers.

To negate a number its two's complement is taken. The definition of the two's complement operation is: "the one's complement plus one". The one's complement is easy -- every bit in the value is reversed. The following table shows these operations.

DEC	BINARY	ONE'S C	TWO'S C	DEC
1	00000001	11111110	11111111	-1
2	00000010	11111101	11111110	-2
17	00010001	11101110	11101111	-17
32	00100000	11011111	11100000	-32

The one's complement plus one rule is somewhat cumbersome to use. A better rule for finding the two's complement is to invert all bits up to but not including the last "one" bit in the value.

As mentioned before, the same instructions operate on both signed and unsigned numbers. Because of the two's complement notation used for signed numbers, the computation and the result are the same. Differences occur only in the interpretation of the result. The program interprets the result not the computer. The program interprets the result of a computation by testing the status bits in the status register. If the calculation was on signed numbers, the program tests the A>, EQ or OV status bits. If the calculation was on unsigned numbers, the program tests the L>, EQ or CA status bits. The meaning of the status bits is shown below.

A>	Arithmetically greater than (signed)
EQ	Equal to (signed)
OV	Overflow, number too large (signed)
L>	Logically greater than (unsigned)
EQ	Equal to (unsigned)
CA	Carry, number too large (unsigned)

The status bits are not set directly by a calculation. They are set only by a comparison of two values, however, after almost every calculation, the result is compared to zero in order to set the status bits. Note that zero (or equal) is the same for either signed or unsigned numbers.

9900 CPU REGISTERS

The 9900 CPU has four "hardware registers" used to control the CPU and maintain its status. The hardware registers are:

```
PC  - Program Counter
WP  - Workspace Pointer
ST  - Status
CRU - Communications Register Unit
```

The 9900 also has multiple sets of "software registers" each set of which is called a workspace. Each workspace consists of 16 consecutive words of memory, giving 16 software registers in a set.

The hardware registers are seldom referred to directly and thus their full name is usually used when they are referred to. The term "register" is usually used to refer to one of the software workspace registers.

Program Counter Register

The Program Counter (PC) contains the word address of the next instruction to be executed by the CPU. As part of the execution of each instruction the PC is updated to point to the next instruction.

Status Register

The Status Register (ST) is a 16-bit register which maintains the status of the computer. Each of the bits has a particular function. These functions are listed below.

BIT	ID	FUNCTION
0	L>	Logical greater than flag
1	A>	Arithmetically greater than flag
2	EQ	Equal to flag
3	CA	Logical carry flag
4	OV	Arithmetic overflow flag
5	OP	Odd parity in byte flag
6	X	XOP instruction flag
7	--	Unused
8	--	Unused
9	--	Unused
10		Unused
11	--	Unused
12-15	--	Interrupt mask

Workspace Pointer Register

The Workspace Pointer register (WP) has the memory address of the current workspace. The workspace registers may be addressed by their register number, 0 to 15, or by their full memory address.

Special Workspace Registers

Some workspace registers have special functions. These special functions are shown below.

REGISTER	FUNCTION
0	Cannot be used for indexing
0	Holds shift count when the count in the shift instruction is zero
11	Contains the return address from the BL instruction
11	Contains the effective address for the XOP instruction
12	Contains the CRU base address in bits 0-14
13	Contains the saved WP during BLWP, RTWP, XOP and interrupts
14	Contains the saved PC during BLWP, RTWP, XOP and interrupts
15	Contains the saved ST during BLWP, RTWP, XOP and interrupts

Each of the above registers can be used for other things when the noted instructions are not being used.

Communications Register Unit

The Communications Register Unit (CRU) is a 4096-bit register that is used by the CPU to communicate with the devices attached to it. Workspace register 12 contains the CRU base address in bits 0-14 when using instructions that test or modify CRU bits. Because the CRU bit address uses only the first 15 bits of register 12, the bit addresses used in programming are two times the hardware address. For example, to load the base address for CRU bit 5 the following would be coded.

```
          LI    R12,10      CRU BASE ADDRESS 5
```

CARRY AND OVERFLOW

The Carry Status Bit (CA) is set when a bit is carried out to the left in arithmetic or shift operations, and as such, represents overflow in unsigned arithmetic. The Overflow Status Bit (OV) is set when signed arithmetic or shift operations cause an overflow. Overflow occurs when the result is too large to represent in the number system being used. Some examples shown below may clarify the setting of the CA and OV status bits. Each example shows the operation done to 20 bits of accuracy so that the carrys and overflows can be seen.

```
20 Bit:    0FFFF  +  0FFFF   = 1FFFE
16 Bit:     FFFF  +   FFFF   =  FFFE
Signed:     -1    +   -1     =  -2       OV=0
Unsigned:  65535  +   65535  =  65534    CA=1

20 Bit:    0FFFF  +  00001   = 10000
16 BIT:     FFFF  +   0001   =  0000
Signed:     -1    +   1      =  0        OV=0
Unsigned:  65535  +   1      =  0        CA=1

20 Bit:    07FFF  +  00001   = 08000
16 Bit:     7FFF  +   0001   =  8000
Signed:    32767  +   1      =  -32768   OV=1
Unsigned:  32767  +   1      =  32768    CA=0

20 Bit:    07FFF  +  07FFF   = 0FFFE
16 Bit:     7FFF  +   7FFF   =  FFFE
Signed:    32767  +   32767  =  -2       OV=1
Unsigned:  32767  +   32767  =  65534    CA=0

20 Bit:    08000  +  08000   = 10000
16 Bit:     8000  +   8000   =  0000
Signed:   -32768  +  -32768  =  0        OV=1
Unsigned:  32768  +   32768  =  0        CA=1
```

Subtract is done as "two's complement and add".

```
16 Bit:     FFFF  -   FFFF   =  0000
20 Bit:    0FFFF  +  00001   = 10000
Signed:     -1    -   -1     =  0        OV=0
Unsigned:  65535  -   65535  =  0        CA=1

16 Bit:     8000  -   8000   =  0000
20 Bit:    08000  +  08000   = 10000
Signed:   -32768  -  -32768  =  0        OV=0
Unsigned:  65535  -   65535  =  0        CA=1

16 Bit:     8000  -   0001   =  7FFF
20 Bit:    08000  +  0FFFF   = 17FFF
Signed:   -32768  -   1      =  32767    OV=1
Unsigned:  32768  -   1      =  32767    CA=1

16 Bit:     0001  -   0001   =  0000
20 Bit:    00001  +  0FFFF   = 10000
Signed:     1     -   1      =  0        OV=0
Unsigned:   1     -   1      =  0        CA=1
```

ADDRESSING MODES

The instructions in a program must address the data in memory or the CRU on which they are to operate. The 9900 provides 7 addressing modes to provide both compactness of code and flexibility in dealing with data structures in memory, and one addressing mode for addressing the CRU bits. The 8 modes are:

1 Direct Memory
2 Indexed Memory
3 Workspace Register Direct
4 Workspace Register Indirect
5 Workspace Register Indirect Auto-increment
6 Immediate Data
7 Program Counter Relative
8 CRU Relative

Modes 1 to 5 are collectively called "general address mode". Modes 6 to 8 are used in instructions which allow only the individual modes. Each of these addressing modes is described in the following sections.

Direct Memory

In this mode of addressing memory, the full 16-bit address of the memory location is contained in the instruction. The length of the instruction is extended to 2 or 3 words as needed. Direct memory addressing is indicated in Assembler language by preceding the symbolic or actual memory address by an at sign (@). For example:

```
CLR   @FLAG          ZERO FLAG WORD
MOV   @A,@B             MOVE A TO B
```

Indexed Memory

In this mode of addressing memory, a 16-bit base address is given in the instruction and an index value in a workspace register is specified. The two 16-bit values are added together to give the full or effective address. Indexed memory addressing is indicated in Assembler language by preceding the symbolic or actual base address by an at sign and following it with the index register number in parentheses. Note that workspace register zero cannot be used for indexing. For example:

```
CLR   @FLAG(R2)    ZERO FLAG INDEXED BY R2
MOV   @A,@B(R3)    MOVE A TO B INDEXED BY R3
```

Workspace Register Direct

In this mode of addressing memory (remember that the workspace registers are in memory), the quantity being addressed is in one of the workspace registers. The register number, 0-15, is specified in the instruction. In Assembler Language, workspace register addressing is assumed if no other indication is given. For example:

```
CLR   R2            ZERO REGISTER 2
MOV   R2,R5         MOVE VALUE IN R2 TO R5
```

Workspace Register Indirect

In this mode of addressing memory, the actual memory address is in a workspace register. The register number containing the address is specified in the instruction. In Assembler Language, workspace register indirect addressing is specified by preceding the register number by an asterisk. For example:

```
LI    R2,>A000     R2=MEMORY ADDRESS
CLR   *R2          CLEAR WORD AT >A000
MOV   R0,*R2       MOVE VALUE IN R0 TO >A000
```

Workspace Register Indirect Autoincrement

In this mode of addressing memory, the actual memory address is in a workspace register just as for workspace register indirect addressing. However, as well as using the address in the register to access memory, the value in the register is incremented after it has been used. The value is incremented by one for byte operations and by two for word operations. The register number containing the address is specified in the instruction. In Assembler Language, workspace register indirect addressing is specified by preceding the register number by an asterisk and following the register number by a plus sign. For example:

```
LI    R2,>A000     R2=MEMORY ADDRESS
CLR   *R2+         CLEAR WORD AT >A000, INC R2
MOV   R0,*R2+      VALUE IN R0 TO >A002, INC R2
```

Immediate Data

In this mode of addressing, the instruction itself contains the word of data to be used. The operation is performed on a value in a workspace register using the data in the instruction. Immediate instructions are all two words long, one word for the instruction and register number and one word for the immediate data. For example:

```
LI    R2,10        R2=10
AI    R2,5         R2=15
```

Program Counter Relative

This addressing mode is used only by the jump instructions. The jump instructions contain a one byte signed displacement which is multiplied by 2 and added to the Program Counter to cause a jump to the new address (if test for the jump is true). Since the displacement is multiplied by two it is actually a signed number of words. Note that the Program Counter will be updated to the address of the next instruction (i.e. the one following the jump) before the displacement is added. In Assembler Language, the jump instructions are coded with a label to indicate the jump target and the Assembler calculates the displacement necessary for the instruction. Note that the range of the jump is limited to -128 words to +127 words from the byte following the jump instruction.

The Assembler will issue a "RANGE ERROR" message is the target is not within this range. For example:

```
        JMP   A             JUMP TO A
        JGT   B             IF GREATER THAN, JUMP TO B
        ....
A       MOV   R2,R3
B       A     R1,R3
```

CRU Relative

The single bit CRU instructions, TB, SBO and SBZ are the only ones to use CRU Relative addressing. The instructions contain a one byte signed displacement. The displacement times two is added to the CRU base address in workspace register 12 to form the full CRU address of the bit being operated on.

CONTEXT SWITCHING

A "context switch" takes place when the CPU begins executing an independent piece of code. During the switch all the CPU's main registers are saved and reloaded with new values. There are several cases where this type of context switch is useful and desirable.

The first case where this is useful, from a programming standpoint, is a call to a separately assembled subroutine. The BLWP instruction performs a context switch designed for subroutine calls. The RTWP instruction reverses the switch and returns back to the calling code.

Another case where a context switch is useful is when calling the operating system to perform some service. The XOP instruction is designed for just such a purpose.

Finally, when an interrupt from the video processor, an I/O device, etc., occurs a context switch occurs.

The same mechanism is used for all these context switches. In all cases, a new WP and a new PC are loaded from two consecutive words of memory and then the old WP, old PC and old ST are saved in the new workspace registers 13, 14 and 15. The two consecutive words of memory that contain the new WP and PC values is called a "context switch vector" or a "transfer vector".

The address of the transfer vector depends upon the cause of the switch. The BLWP instruction contains the address of its transfer vector. The XOP instruction contains a code value from 0 to 15 which is used to select one of the 16 transfer vectors that are located in memory beginning at address >0040. As well as storing the WP, PC and ST, the XOP also stores in the new workspace

register 11 the effective address developed for the general address specified in the instruction.

The type of interrupt being processed is used to select the transfer vector from fixed addresses in memory.

INT LEVEL	VECTOR ADDR	INTERRUPT SOURCE
0	>0000	External RESET
1	>0004	External
*	>FFFC	External, non-maskable

During interruption the ST is set to a value which will mask all lower priority interrupts if they can be masked.

In all the context switches, a return to the previous context is done with the RTWP instruction which restores the saved hardware registers from workspace registers 13, 14 and 15.

AEMS MAPPER

The AEMS card uses the Texas Instruments SN74LS612 memory mapper chip along with additional logic to map the 32K memory addresses from a 16 bit address to a 24 bit address. A 24 bit address accommodates a 16 Megabyte memory.

This mapping is done by splitting the TI 99/4A 16 bit address into two parts: a 4 bit page number and a 12 bit page offset. The 12 bit page offset gives a 4K page. The 4 bit page number is used to select a "mapper register" containing a 12 bit extended page number. The 12 bit extended page number is combined with the original 12 bit page offset to give a 24 bit expanded memory address.

The mapper can be inactive or active. When the mapper is inactive the TI 99/4A will operate as though it had an ordinary 32K memory card. At power on, the mapper is inactive.

The mapper is activated by setting a CRU bit (>1E04) to one. When active, only addresses >2000 to >3FFF and >A000 to >FFFF are mapped. Mapping can be turned off by setting the CRU bit to zero.

The mapper has 8 active "registers" that specify which pages are mapped into the TI 99/4A's address space. In order to access these registers (for write or read) the access must be enabled by setting CRU bit >1E02 to one. Access is disabled by setting the CRU bit to zero. The mapper registers are accessed as 16-bit values by normal 9900 instructions at the following addresses:

Register Number	Access at Address	Maps Page in at
2	>4004	>2000
3	>4006	>3000
A	>4014	>A000
B	>4016	>B000
C	>4018	>C000
D	>401A	>D000
E	>401C	>E000
F	>40F0	>F000

The access addresses will respond to instructions just like a normal word of RAM.

Typical code for operation of the mapper is shown below.

```
* Load Mapper Registers
      LI    R12,>1E02     CRU base address
      SBO   0             Enable register access
      LI    R0,20         Page # 20
      MOV   R0,@>4014     Map page in at >A000
      LI    R0,50         Page # 50
      MOV   R0,@>4016     Map page in at >B000
      SBZ   0             Disable register access
*
      SBO   1             Turn mapper on
      MOV   @>A000,R0     Get 1st word page 20
      MOV   @>B002,R1     Get 2nd word page 50
      SBZ   1             Turn mapper off
```