MMP16

A 16-bit Didactic Micro-Programmed Micro-Processor

José Luis López Presa
Department of Telematic Engineering and Architecture
Polytechnic University Madrid
Madrid, Spain
jllopez@diatel.upm.es

Elio Pérez Calle
EU Science and Technology Fellowship Programme
University of Science and Technology of China
Hefei, Anhui, China
elio@ustc.edu.cn

Abstract— **MMP16 stands for 16-bit Didactic Micro-Programmed Micro-Processor and consists in a comprehensive learning tool for those students of electric engineering and related disciplines that, having studied the basics of digital electronics, need to understand the principles of computer organization prior to study advanced computer architecture. Therefore, MMP16 has been designed as a bridge between the study and use of digital electronic components and a full implementation of a digital microprocessor. MMP16 is a powerful tool not only to depict the basic concepts of a microprocessor layout and operation, but also to instruct advanced concepts such as pipelining. Besides the theoretical approach, MMP16 may be used as a software developing tool, allowing the students to develop and test their own microprograms using the simulation software provided. MMP16 is currently being used in the Universidad Politécnica de Madrid (UPM) and its deployment is under study in several European and Chinese universities.**

Keywords- Principles of Computer Organization, Computer Architecture, Computer Simulation, Didactic Tools.

I. MOTIVATION

As some authors point out [1], although today most students start undergraduate courses having already had contact with computers as users, computer organization and architecture involves many abstract concepts for those undergraduate beginners. Therefore, most of them might understand the computer as the tool they use for Internet browsing and social networking on a daily basis. In this situation, professors are required to fill an enormous gap between the end-user approach of students and the principles of computer organization, whose concepts are completely new to them.

This gap is caused by the increasing number of abstraction layers interposed between the hardware and end-user oriented applications. A depiction of this can be found in [2], where Tanenbaum defines a 6-tier structure present in modern computers[1], going from the digital logic level to the problem-oriented language level, as shown in Fig. 1. MMP16 is focused on the second level of this hierarchy (microprogramming level) and requires the students to have basic knowledge of circuit theory and elemental digital components, such as logic gates, multiplexers, decoders, adders, registers, and so on, as defined in most electrical and electronic engineering and computer science and engineering undergraduate curricula.

The motivation behind the design of MMP16 is to link digital electronics with computer architecture and, therefore, show the students how a computer can be built using the simple components they already know.

Furthermore, the use of a working example is the best way to understand the key concepts of computer organization. Based on the excellent theoretical approach developed by some authors [2] [3] [4], MMP16 provides not only the design of a microprogrammed microprocessor aimed at learning purposes, but also non-commercial software to be used in laboratory classes. This approach overcomes the limitation of available commercial software, and particularly the time-consuming learning of the program interface by the students and the orientation towards professional applications [5] [6] [7]. On the contrary, MMP16 is completely focused on learning, in both theoretical and practical skills.

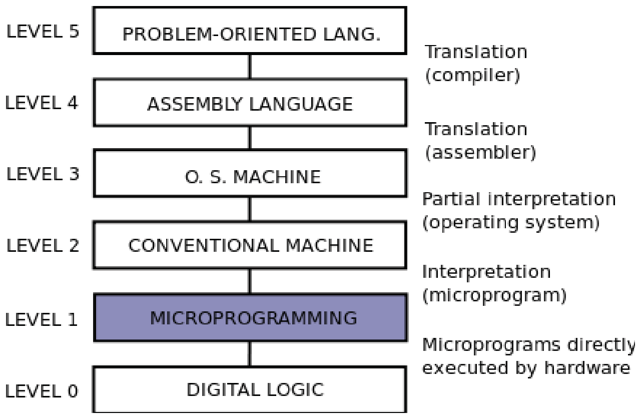

Figure 1: Computing levels defined by Prof. Andrew Tanenbaum.

1 Nowadays, microprogramming is not as popular as it used to be, since RISC computers do not need a sophisticated control unit.

978-1-61284-840-2/11/$26.00 ©2011 IEEE

II. Design

MMP16 has been designed with a strong emphasis on the principles of computer organization, and therefore the processor is neither complex nor modern. Nevertheless, the model is complex enough to allow the student to learn the basics of the design of a simple processor, and to introduce them to advanced concepts such as pipelining. The design is focused on the CPU, as the main goal of MMP16 is to consolidate the student´s knowledge about CPU inner organization and instruction sequencing. MMP16 covers the main principles of computer organization and provides the students enough understanding of the structure of a computer to be able to tackle more advanced concepts and techniques such as the massive use of pipelining, parallel processing, etc.

Fig. 2 shows the full simulated system formed by MMP16 and a 64K 16-bit memory. In order to achieve the highest simplicity, a byte word cannot be addressed. Every memory address is a 16-bit word and both the address and the data buses are also 16-bit wide, following the design pattern that makes 16-bit the standard for this processor.

MMP16 has the following external signals: *r*, indicating reading of a memory position; *w*, indicating writing; *ready*, that is used by the memory to indicate the processor that the memory operation has concluded; and *reset*, that allows to reset the machine. It loads the fixed 0xFFFE address into the Program Counter and starts a FETCH cycle. At that memory address, a jump instruction is expected to be found. A ROM coded program detailing startup instructions should be stored here –usually the BIOS program, that would access the hard disk lo load the operating system or perform a network start. There is also an interruption petition line *(int)*, that can be used by Input/Output (IO) devices, though in this version of MMP16, no IO devices are included. MMP16 is designed to use memory mapped IO, hence it does not need any extra signals for IO operations.

Like any processor, MMP16 has a Control Unit, and a Data Processing Unit. MMP16 uses two different clock signals: φ1 is used for the Control Unit and defines the machine states, and φ2 is used to load the registers of the Data Processing Unit and synchronize the memory signals and buses.

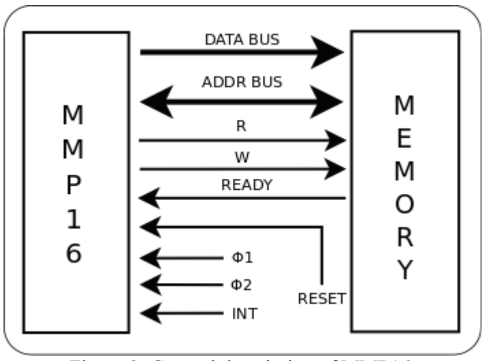

Figure 2: General description of MMP16.

III. Organization And Architecture

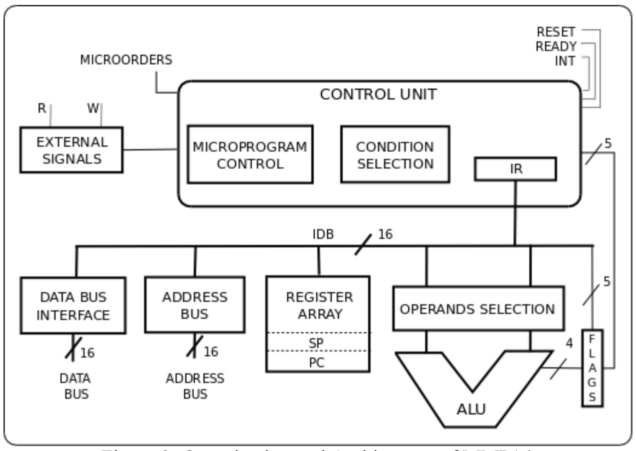

Figure 3: Organization and Architecture of MMP16.

Fig. 3 shows the block diagram of MMP16. The organization is based on an internal data bus (IDB) interconnecting the functional blocks of MMP16. For the sake of simplicity, the registers of the Data Processing Unit and the IDB have 16 bits.

The fact that MMP16 is not able to address bytes (commonly used to represent characters) may be considered an important limitation, but nowadays it is very common to work with 16-bit charsets, such as the Guobiao system for Chinese characters.

MMP16 has a register array of 16 registers (R0, R15). R14 is the stack pointer (SP) and R15 is the program counter (PC), as shown in Fig. 4. Registers may be addressed in two ways: using directly some operand fields of the Instruction Register (IR), or using specific microorders generated by the control unit (used to address SP and PC). In order to accelerate PC and SP increment and decrement operations, specific hardware is provided within the register block, avoiding the use of the ALU for these simple operations.

MMP16 is microprogrammed. Its control memory consists of 256 32-bit words. To address the control memory, 8-bit microaddresses are required. Each of these words will contain a microinstruction (MMP16 has a reduced instruction set, but it is not RISC). Again, a microprogrammed control unit has been chosen instead of a wired one for the sake of simplicity.

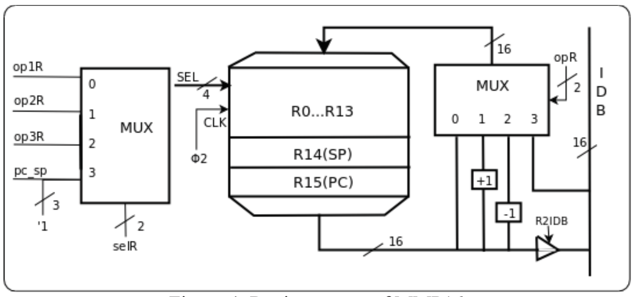

Figure 4: Register array of MMP16.

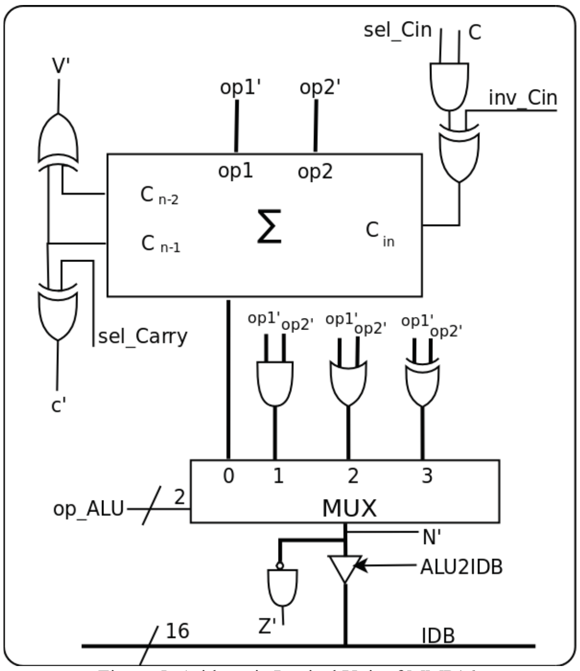

Figure 5: Arithmetic Logical Unit of MMP16.

The 16-bit Arithmetic and Logical Unit (ALU), shown in Fig. 5 is able to perform arithmetic operations such as addition, subtraction and logic operations such as AND, OR, XOR and NOT.

IR can be loaded from IDB activating the *ld_IR* signal. The format of the instructions of MMP16 has been designed to simplify its decoding. Some fields of IR are used in different areas of the system with different names, as shown in Table 1.

The format of the available instructions in the IR is detailed in Table 2. Their operation code can occupy 4, 8, 12 or 16 bits. The code always occupies the most significative bits of the instructions first word. 16-bit codes have their 12 MSB (cop1, cop2 and cop3) to 1. 12-bit codes have their 8 MSB (cop1 and cop2) to 1. 8-bit codes have their 4 MSB (cop1) to 1. This allows up to 15 4-bit codes, 15 8-bit codes, 15 12-bit codes and 16 16-bit codes. 45 different codes have been selected out of the 61 possible ones.

TABLE I. FORMAT OF THE INSTRUCTION REGISTER (IR)

Instruction Registry (IR)			
cop1	op1R	op2R	op3R
	cop2	cop3	cond
	immediate		cop4
15 14 13 12	11 10 09 08	07 06 05 04	03 02 01 00

The second word of a two-word instruction always contains an immediate data o a data that is used to calculate the effective address of the data in the memory.

TABLE II. CODES OF OPERATION AND UBICATION OF THE CORRESPONDING MICROPROGRAM.

cop1	cop2	cop3	cop4	address
XXXX = F	---	---	---	1XXXX000
F	XXXX = F	---	---	0XXXX000
F	F	XXXX = F	---	0XXXX100
F	F	F	XXXX	1XXXX100

IV. PIPELINING

MMP16 is organized as a two segment pipeline, as shown in Fig. 7. The pipeline is controlled by clock signal φ1. The first segment is in charge of fetching the next microinstruction, while the second one controls the operations on the Data Processing Unit. Segmentation is implemented by the use of the state register and the microinstruction register. The first segment starts from the state register, goes through the condition selection logic, then the microprogram counter that decides which will be the next microinstruction and the control memory that stores microprograms. It finishes loading the microinstruction into the microinstruction register.

The second segment starts from the microinstruction register that distributes the microorders to control the different blocks of the Data Processing Unit: registers, ALU, bus interfaces, etc. When φ2 arrives, the registers of the Data Processing Unit are loaded. Then, the new state is loaded into the status register to control the activity of the first segment during the next cycle.

This kind of pipeline is one of the most frequent ones. Its aim is to parallelize two slow processes: the search of the next microinstruction in the memory and the execution of the current microinstruction. The basic characteristic of this kind of organization is that the election of the next microinstruction, when a conditional jump occurs in the microprogram, is made depending on the state of the machine at the end of the previous microinstruction and not depending on the state of the machine at the end of the current cycle.

V. PROGRAMMING

One of the main features of MMP16 is the possibility for the students to code and test their own microprograms. Writing microprograms and testing them using MMP16's software simulator allows the students to better understand the inner operation of the processor.

```
;op2_0=0, inv_op2=1, op_ALU=OR, ALU->PC;
01111100: 10xxxx011111100x000000xx111xx0x0
;selR=pc_sp, pc_sp=PC, decR, sel_f=2, mod_f;
01111101: xxxxxxxx11110xxx00000110111xx0x0
```

Figure 6: Reset Microprogram

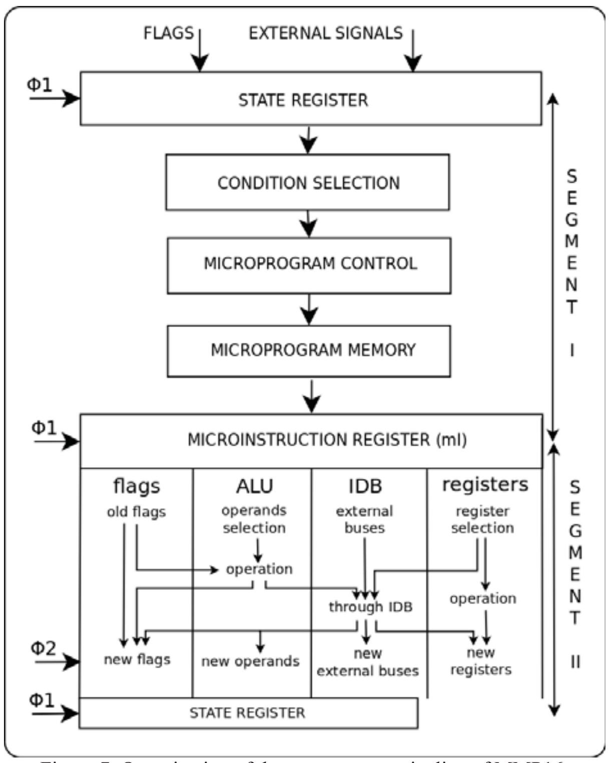

Figure 7: Organization of the two-segment pipeline of MMP16.

Fig. 6 and 9 are provided as an example. MMP16 starts with a RESET, that executes the microinstruction at address 0x8C. Here the system initialization should be done. The program counter is set to 0xFFFE and interrupts are disabled. At address 0xFFFE, there should be a jump instruction to the beginning of the system initialization program. All this should be stored in ROM in a real computer system.

After the RESET procedure, MMP16 starts a FETCH cycle. First, it is checked if there is a pending interrupt to be processed. If there is, the flags register and the current PC has to be stored in the stack, the content of main memory address 0 is loaded into the PC[2]. and the interrupts are disabled. Then, a memory read cycle will be performed to fetch the instruction the PC points at, followed by a jump to the microprogram that interprets that instruction, which ends jumping to the FETCH routine. This is a simple example of the microcode for the MMP16. All the details can be found at the MMP16 website.

The set of instructions of MMP16 can be divided in six different subsets: arithmetic and logic instructions, load instructions, store instructions, sequence breaking instructions, stack management instructions and miscelaneous (those that cannot be classified in any of the former groups). Instructions can occupy one or two memory words (16 or 32 bits). The set of instructions is depicted in Fig. 8.

Arithmetic and logic instructions (16bit)

AND op1R, op2R, op3R	0011	op1R	op2R	op3R
OR op1R, op2R, op3R	0100	op1R	op2R	op3R
XOR op1R, op2R, op3R	0101	op1R	op2R	op3R
NOT op2R, op3R	1111	0100	op2R	op3R
CMP op2R, op3R	1111	1101	op2R	op3R
ADD op1R, op2R, op3R	0110	op1R	op2R	op3R
SUB op1R, op2R, op3R	0111	op1R	op2R	op3R
ADC op1R, op2R, op3R	1000	op1R	op2R	op3R
SBB op1R, op2R, op3R	1001	op1R	op2R	op3R
ADI op3R, inmediate	1010	inmediate		op3R
SBI op3R, inmediate	1110	inmediate		op3R

Load instructions (16bit and 32bit)

LIM op3R, inmediate	1111	1111	1010	op3R	inmediate
LDD op3R, address	1111	1111	0100	op3R	address
LDR op3R, base[op2R]	1111	0000	op2R	op3R	base address
LDR op3R, op2R[offset]	1111	0000	op2R	op3R	offset
LDX op3R, op1R[op2R]	1011	op1R	op2R	op3R	
LDI op3R, [op2R]	1111	1110	op2R	op3R	

Store instructions (16bit and 32bit)

STD op3R, address	1111	1111	0101	op3R	address
STR op3R, base[op2R]	1111	0001	op2R	op3R	base address
STR op3R, op2R [offset]	1111	0001	op2R	op3R	offset
STX op3R, op1R[op2R]	0010	op1R	op2R	op3R	
STI op3R, [op2R]	1111	0010	op2R	op3R	

Sequence breaking instructions (16bit and 32bit)

JMP cond, address	1111	1111	0110	cond	jump add
CALL cond, address	1111	1111	0111	cond	add
JMPI cond, [op2R]	1111	1011	op2R	cond	
CALLI cond, [op2R]	1111	0011	op2R	cond	
JMPR cond, desp	1100	offset/inm		cond	
CALLR cond, desp	1101	offset/inm		cond	
RET cond	1111	1111	1001	cond	
RETI cond	1111	1111	1110	cond	

Stack management instructions (16bit)

PUSH op3R	1111	1111	1000	op3R
POP op3R	1111	1111	1011	op3R
PUSH FLAGS	1111	1111	1111	1110
POP FLAGS	1111	1111	1111	1100

Miscelaneous instructions (16bit)

CLRI	1111	1111	1111	1010
SETI	1111	1111	1111	1001
MOV op2R,op3R	1111	1100	op2R	op3R

Figure 8: Set of instructions of MMP16.

VI. SIMULATION

The operation of MMP16 can be simulated using a free software developed by the authors and available for both Windows and GNU/Linux platforms from the official website of the program. Using this simulator, the students can test if their microprograms execute properly the selected instructions.

The simulator includes commands to store the desired values in the control memory, the core memory of the simulated system and any register in the CPU, or print their values, at any time. The processor may be run some number of clock cycles of both φ1 and φ2. The state of the simulation may be saved and restored, so it can be resumed in the future.

2 Usually, the interrupt vector is stored at the beginning of the memory space.

```
;jmp 10000111 if not int, save_mpc;
01111110: 10000111xxx00xxx000000xx101101x0
;selR=pc_sp, pc_sp=SP, decR, flags->data;
01111111: xxxxxxxx0111010x001000xx111xx0x0
;selR=pc_sp, pc_sp=SP, decR, Registers->addr, w;
10000000: xxxxxxxx0111011x000100xx111xx001
;jmp $ if not ready;
10000001: 10000001xxx00xxx000000xx011100x0
;PC->data, sel_f=block_int, mod_f;
10000010: xxxxxxxx1110011x00100110111xx0x0
;selR=pc_sp, pc_sp=SP, Registers->addr, w;
10000011: xxxxxxxx0110011x000100xx111xx001
;jmp $ if not ready;
10000100: 10000100xxx00xxx000000xx011100x0
;op2_0=0, inv_op2=0, op_ALU=AND, ALU->addr, r;
10000101: 01xxxx00xxx0000x000100xx111xx011
;data_bus->PC, jmp $ if not ready;
10000110: 1000011011111101x000000xx011100x0
;selR=pc_sp, pc_sp=PC, incR, Registers->addr, r;
10000111: xxxxxxxx1110111x000100xx111xx011
;data_bus->IR, jmp $ if not ready;
10001000: 10001000xxx0001x000010xx011100x0
;jmp inst;
10001001: xxxxxxxxxxx00xxx000000xx110110x0
```

Figure 9: Fetch microprogram.

A typical session would include the following steps:

1. Load the microprograms into the control memory.
2. Load the program into main memory.
3. Send a reset pulse.
4. Run the processor some clock cycles.
5. Test the content of the desired registers or memory words.

The simulator is the perfect complement to the practical exercises in order to understand instruction sequencing and the data path management, and helps the students to comprehend the inner operation of the processor they have already studied in theory. Furthermore, MMP16's software simulator is the last step in the process of connecting the student's knowledge about digital electronics (that can be applied to understand MMP16's hardware implementation) to the principles of computer organization and the basics of computer architecture (whose operation can be better understood thanks to the simulator).

VII. CONCLUSIONS

MMP16 covers the gap between digital electronics and computer architecture offering a complete set of didactic tools: the design of a modern computer, the practical application of the principles of computer organization including advanced concepts such as pipelining, the possibility to code and test microprograms, and finally a software simulator as the last step of the process.

The use of MMP16 as a didactic tool has improved the results of the students that learn principles of computer architecture at UPM. Based on the obtained results, we regard as the main advantages of this didactic tool its utility to understand key concepts despite its simplicity, the existence of a simulator that can be used to test their own microprograms and therefore connect theory and practice, and the link that MMP16 provides between digital electronic and computer architecture. The authors expect to extend the use of MMP16 to other higher education institutions in the nearer future. A document with the full description of the MMP16 with the full instruction set as well as the software simulator and other teaching materials can be downloaded form its official site[3].

ACKNOWLEDGMENT

This work was partially supported by the EU Science and Technology Fellowship Programme China.

REFERENCES

[1] Calazans, N. L. V. and Moraes, F. G. (2001). Integrating the teaching of computer organization and architecture with digital hardware design early in undergraduate courses. IEEE Transactions on Educations,44(2):109–119.

[2] Tanenbaum, A. S. (2009). Structured Computer Organization. Pearson International.

Tomek, I. (1990). The Foundations of Computer Architecture and organization. W. H. Freeman and Co.

[4] Patterson, D. A. and Hennessy, J. L. (2008). Computer Organization and Design. Morgan Kaufmann.

[5] L. Rodriguez Pardo, M.J. Moure, M. V. and Mandado, E.(1998). Viscp: a virtual instrumentation and cad tool for electronic. In Conference on Frontiers in Education.

[6] Maurer, P. (1998a). Electrical design automation: An essential part of computer engineers education. In Conference on Frontiers in Education.

[7] Maurer, P. (1998b). Enhancing the hardware design experience for computer engineers. In Conference on Frontiers in Education.

3 http://www.diatel.upm.es/jllopez/mmp16/