TEACHING ASSEMBLY LANGUAGE USING AN ASSEMBLY LANGUAGE INTERPRETER[1]

Trevor Mudge
Department of Electrical and Computer Engineering
University of Michigan
Ann Arbor, MI 48109

Summary

This paper outlines the background of teaching needs that led to the development of an assembly language interpreter for teaching assembly language. The interpreter, called ZIP for Z80 interpreter program, is small enough to be used by a student on a modest sized Z80 based system. It greatly facilitates understanding the Z80 assembly language by allowing the student to learn it interactively. It can also be used as a debugging tool. An overview of the interpreter is given; its syntax and operation are discussed.

1. Introduction

A sequence of three courses is being offered by the Electrical and Computer Engineering (ECE) Department and the graduate program in Computer, Information and Control Engineering (CICE) at the University of Michigan. to educate students in the use of microprocessor-based digital systems design [1]. Design is taught in the context of microprocessors and other related very large scale integrated (VLSI) components, including ROM and RAM memory chips, PLAs, timers, sequencers, PIOs, SIOs, multipliers, and floating point attached processors. These VLSI components are taken principally from the Zilog Z80 family, the Advanced Micro Devices Am2900 family, and the Intel 8086 family. The three courses are: ECE 365—Digital Computer Engineering, ECE 366—Digital Computer Engineering Laboratory, and ECE 466—Digital Design Laboratory. ECE 365 is intended for juniors, ECE 366 for seniors, and ECE 466 for seniors or first year graduate students. These courses have been substantially upgraded from their original form, and more sophisticated material has been included. This has made the courses considerably more difficult for students, particularly at the entry level--ECE 365— where they are first exposed to assembly language. To reduce this difficulty several teaching aids are being developed. Principal among these is ZIP, a Z80 interpreter program. An alternative solution to reducing the difficulty of the courses would be to spread the course load over more courses, however the need to keep faculty teaching loads to a reasonable level precluded this approach.

The following few paragraphs outline the history of the three courses, how they have been continually upgraded to reflect developments in the digital systems design field, and by implication how this has led to a much more demanding teaching and learning environment. In particular, one that needs teaching aids such as ZIP.

2. Background

Originally, in the period 1968 through 1978, 365 and 366 were taught using five PDP8 minicomputers—four for ECE 365 and one for 366-- and a number of attache case "logic lab" kits that contain a 5 volt power supply, push buttons, switches, built-in sockets for ICs, wire, wire strippers, pliers, logic probes, and a selection of TTL logic. Assembly programming and logic design were introduced in 365, and I/O programming and projects involving interfacing hardware to PDP8s were introduced in 366.

During the first part of the same period 466 was an infrequently taught lab course in advanced digital design projects. In the mid 1970s 466 was revived and updated to include projects involving PDP16 Register Transfer Modules (RTMs) as well as the then new Intel SIM8-01 system—an Intel 8008 based single board computer [2]. Furthermore, a link was established to MTS (Michigan Terminal System—the campus wide computing facility) where utilities such as cross-assemblers, editors, etc., were available. This allowed rapid software development outside the lab for projects involving RTMs or the SIM8-01. This reduced the time needed in the lab, allowing more efficient use of the lab resources. The success of this link prompted the development of similar facilities for 365 and 366.

By 1977 the demand for 365 and 366 had grown, and both courses, which still relied on PDP8s, were very much outdated. It was felt that the courses needed updating and that modern microprocessor-based equipment was needed in their labs. Therefore, a proposal [3] to fund new lab equipment was submitted to NSF under their Instructional Scientific Equipment Program. It was successful, and together with matching funds from the University of Michigan put about $29,000

[1] This work was supported in part by a grant from the University of Michigan's Center for Research on Learning and Teaching.

at our disposal to develop a "Microprocessor Based Computer and Digital Systems Design Laboratory". The proposal called for 12 lab stations to be built by us and shared by both 365 and 366. Each station was to be constructed around a Motorola MEK 6800D2 kit, a 16Kx8 bit RAM board, a low cost CRT-keyboard terminal, and a cassette tape recorder. These lab stations were to be used in conjunction with the logic lab kits. However, a gift of $70,000 from the XYCOM Corporation of Saline, Michigan allowed us to acquire ready made microprocessor development systems for the lab stations to be used in conjunction with the logic lab kits. As a result, 10 of XYCOM's Z80 based development systems--RacPac 3905s--were purchased and introduced into 365 and 366 during 1979. In addition, three printers were obtained--two Okidatas and one DEC Centronics. We considered the RacPacs to be a better choice for lab stations than our proposed home built unit as they had much more capability (see later), they were tried and tested, they were in industrial quality packages, and they came complete with the normal complement of software for a development system. These advantages allowed us to get the labs operational much more rapidly than would have been the case if we had had to stay with our original proposal. Furthermore, additional funds were recently provided by the ECE Department to purchase two more RacPacs in the fall semester 1980.

In 1979 it was felt that 466 had also become dated. Therefore, funds were raised within the department to develop four state-of-the-art lab stations for 466. Building the stations was begun by 466 students as an in-class project. Hardware development was completed during the winter semester 1980. Hardware validation and software development are are in progress. The stations are based on the Intel SDK86 evaluation kit, which includes an 8086 microprocessor, and Am2900 bit slice components. We have termed the stations 29/86 systems. Larger logic lab kits have also been built to be used in conjunction with the 29/86 systems.

Table 1 (tables and figures are at the end of the paper) shows how enrollments have grown in the last five years and the projected enrollments for the next three years. In spite of this growth we have actually been able to increase the number of hours each student has in contact with a lab station in both 365 and 366. This was achieved by increasing the number of stations from the PDP8 days, and by opening the lab for longer hours. In the case of 365 and 366 a single lab is used containing all 12 RacPacs. The lab is open 80 hours/week allowing each student in both classes about 4.5 hours/week contact time. This is up from the 1.5 hours allowed when the PDP8s were used.

The next section gives more details about the course that makes the heaviest use of ZIP. The RacPac system on which ZIP runs is also outlined.

3. Digital Computer Engineering--ECE 365

As noted earlier this course is intended as a foundation course for students in the area of microprocessor-based digital system design as well as a service course for those students whose main focus of interest is in other areas of electrical engineering. This last requirement means the course has to be self contained. There are four hours of classes/week not including labs. The course class material centers on digital computer organization, assembly language, and logic design, with particular reference to microprocessor-based systems. Concepts in the classes are reinforced by a series of eight carefully chosen lab experiments that use the RacPac systems and logic lab kits. The course lecture material is in contrast to a more traditional course on digital system design in two important ways:

First, emphasis is placed on using microprocessors and other VLSI components such as PLAs, ROMs, RAMs, PIOs, SIOs, and timers as system components. Gate level logic design is discussed in much less depth than is usual in a more traditional treatment. Topics such as minimizing of combinational logic and state reduction in sequential machines are only briefly mentioned, whereas design techniques that use PLAs and ROMs are emphasized.

Second, not only is assembly language taught but so is the use of the RacPac operating system, the editor, loader, and other utility programs used in system development on the RacPac. This represents a considerable increase in the amount of software taught compared to that taught in a traditional treatment of digital system design. Our experience so far has shown this extra software to be an obstacle to covering the range of material that we feel constitutes a minimum for such a course. Assembly language appears to give the most difficulty to students, therefore, we have started to develop some teaching aids to help students learn assembly language more rapidly. Chief among these is ZIP. Students get a copy of this interpreter for use as a self-teaching aid and as a debugging tool. Development of ZIP has been funded by the University of Michigan's Center for Research on Learning and Teaching. At the time of writing Version 1 is in use.

ZIP presently runs on the RacPac systems. Each system comprises a CRT, a keyboard, dual single density Shugart 8" disk drives, an RS232 interface, and connections for a printer and an EPROM burner. The logic is built on two boards designated 3744 and 3745. The 3744 (CPU) board contains the Z80 CPU, 32Kx1 bytes of dynamic RAM, 4Kx1 byte EPROM containing the bootstrap loader, two parallel input/output (PIO) chips, two serial input/output (SIO) chips, a counter/timer chip (CTC), a CRT controller (CRTC) chip, and a 2Kx1 byte static RAM to hold the CRT screen. The 3745 (expansion) board contains 64Kx1 byte dynamic RAM, a floppy disk controller chip, EPROM burner control logic, and parallel printer control logic.

In the next section an outline of ZIP's syntax and its operation will be presented.

4. Overview of ZIP

ZIP disassembles specified segments of memory and displays the corresponding code with symbolic operation codes and absolute addresses. Interpretation of the code can be performed one instruction at a time, or in normal sequence until a specific "trigger" condition is met. The instruction to be interpreted next is displayed in inverse video to distinguish it from the other code displayed. The CPU registers are always displayed along with the condition code flags. When an instruction is interpreted any CPU registers or flags that are changed are displayed in inverse video to distinguish them from the unchanged ones. Similarly, certain memory locations that are changed are also displayed in inverse video (see later). Interpreting single instructions clarifies their operation for the novice. Interpreting a program enables the user to relate the program's text (static) with the program's execution (dynamic). This is probably the single most difficult relationship that the novice must learn to visualize. The "trigger" concept, which was borrowed from logic analyzers, makes the interpreter a particularly powerful debugging tool. Blocks of code can be executed until a trigger condition, or a simple logical combination of trigger conditions becomes true. The conditions are specific values of, or relations between, memory locations, registers, or flags.

The layout of the screen is shown in Figure 1. The column on the left shows memory locations in hex (4 hex digits). Alongside these are one to four byte instruction codes also in hex (Z80 instructions can be from one to four bytes in length). Further to the right, the instruction codes are shown in their disassembled form. For example, consider the line covered by the shaded rectangle in the left center. At the left is a memory location (90EE hex). This location and the subsequent one contain the bytes 10 hex and F7 hex respectively (the Z80 has byte oriented addressing). These disassemble to the Z80 instruction "DJNZ 90E7"-- decrement register B and jump if B is non- zero to location 90E7. Notice that addresses of operands or targets of jumps are not disassembled but are left as absolute addresses. To disassemble these would require access to the symbol table created when the program was assembled. In order to keep the first version of ZIP simple the ability to recover symbolic addresses was omitted.

ZIP automatically determines data areas in memory by examining jumps, subroutine calls and returns, and when necessary their targets. Memory locations that contain data rather than instructions have their contents displayed as two hex digits in the same column as the symbolic instruction codes.

The right hand side of the screen displays the contents of the Z80's CPU registers, the top four items on the stack, thirty two bytes of memory, and the command line.

There are eight 1 byte CPU registers: A, F, B, C, D, E, H, L. These are displayed at the top right of the screen. For example, the second row at the top right shows the contents of A in hex (88), the contents of F in hex (33), followed by the contents of A in binary (10001000) and the contents of F in binary (00110011). The binary display is useful for checking bit operations, shifts, and rotations. The F register is not a general purpose register, instead it holds six 1 bit flags that show condition codes. Their position is shown in the binary display of F by the header "SZ*H*PNC" at the extreme top right (see [4] for their meaning). Immediately below the 1 byte CPU registers are the 16 bit CPU registers: IX, IY, SP, PC. IX and IY are index register, SP is the stack pointer (points to top of stack), and PC is the program counter. The register pairs BC, DE and HL can also be regarded as 16 bit registers and the format of the display has been set up to allow this view. To the right of the 16 bit registers appears the two special 1 byte registers I and R. Below the 16 bit registers appears the top four stack items. These items are one word, or two bytes each, thus in Figure 1, for example, the top of stack is at location F3F8 hex (see contents of SP) and the top item is the 16 bit quantity 08ED. The bytes of the top four words of stack are shown the order in which they appear in memory; left-to-right corresponds to low-to-high memory addresses. The stack grows towards low memory. The apparent reversal of the bytes in each word is accounted for by the Z80 convention that words are stored with their most significant byte at the higher memory location.

Below the stack display a user selected 32 byte area of memory is displayed in hex. Finally, below that the command currently being entered by the user is shown.

The shaded rectangles in Figure 1 indicate inverse video. Thus the instruction to be executed next is the DJNZ mentioned above. In addition, the contents of the H and L registers are shown in inverse video indicating that the most recently executed instruction--"INC HL"--caused their contents to be altered. If any of the memory locations already displayed on the screen had been altered they would also be shown in inverse video. In theory, the contents of the R register should be shown as changing at each step because R is a counter bumped during each instruction execution, that is used to form the refresh address. To avoid distraction R is never shown in inverse video.

The command line is displayed with an inverse video square alongside it to distinguish it. The particular command shown in Figure 1 reads: beginning with the current instruction (the DJNZ) execute the program until the contents of register A and B have been equal three times. The command is terminated with a carriage return; the return initiates ZIP's interpretation of the command line. The display scrolls so that the next instruction to be interpreted (i.e. the instruction displayed in inverse video) is always kept in the middle of the screen.

Figure 2 shows the syntax of ZIP's command structure in standard BNF notation. Non-terminal symbols are shown bracketed by "<" and ">". Rewriting rules are identified by "::=" ; the lefthand side can be rewritten as one of the alternatives on the righthand side. The alternatives are separated by "|". The remaining symbols are the terminal symbols that appear in the commands. Spaces can be used freely to aid readability. The syntax is intentionally simple, and in fact can be parsed by a finite state machine. Simplicity was a prerequisite for two reasons. First, size was an issue; we eventually want to be able to use it in smaller systems. Second, the program was to be written by two students with limited programming experience.

When used in conjunction with the RacPac systems ZIP is loaded and run from a floppy disk operating system. Execution begins by querying the user with:

"Start Address?"

The user should respond with a 16 bit hex address indicating where in memory the interpretation should begin. The Z80 allows computed jumps (e.g. JP (IX), see [4]). Version 1 of ZIP cannot handle these. If any of these are present in the region of memory to be interpreted a further message is displayed as follows:

"You are using Computed Jumps!"

If this is the case the interpreter may try to disassemble areas in memory that are data areas. This is not critical so the user is allowed to proceed if he/she wants to. At this point ZIP is ready to receive any of the commands of Figure 2.

The first production in Figure 2 gives the major command classes. The first of these transfers interpretation to a specific location. For example:

GO 10B4

transfers the flow of interpretation to location 10B4 hex. The next alternative allows the modification of registers or memory locations. For example:

SE IX=99AE

sets the contents of the IX register to 99AE hex. (See the second production "<modify> ::=" for other alternatives.) The third alternative of the first production specifies which 32 bytes of memory appear on the screen display. For example:

DI 4F68

displays the 32 bytes beginning with the byte at location 4F68 hex. The fourth alternative to the first production identifies the most important class of commands. They define the trigger conditions. For example:

B = @A77D

causes interpretation until the contents of register B equals the contents of location A77D in memory. Or for example:

F = 0XX1X1

causes interpretation until the flags S=0, P=1, and C=1 (the other three can be either 0 or 1). The simpler forms of this class of commands specify interpretation of the next fixed number of instructions, for example:

10 <u>return</u>

causes the next 10 instructions to be interpreted (<u>return</u> indicates a carriage return). The simplest form is just a carriage return—interpret the next instruction. (See the fourth production "<trigger_condition> ::=" for more details.) The command ON turns on a large display of memory locations that temporarily fill the screen with a memory map beginning with the 32 bytes of memory usually displayed in the normal screen format. The command OF turns it off, reverting back to the normal format. The command AU causes the auxiliary register set to be displayed. The command UA causes the normal register set to be displayed. The command OL causes the old values of the registers—as they were when the last trigger condition was satisfied—to be displayed. The command N causes the new (current) values of the registers to be displayed. Finally, the command QU quits the interpreter.

5. Conclusion

ZIP, an assembly language interpreter, has been outlined. The circumstances that led to its development have also been described. Version 1 of ZIP is running and in use in the lab. In addition, we plan to use the interpreter in conjunction with in-class CRT monitors to explain instructions and to illustrate assembly language concepts.

Several enhancements are planned for Version 2. These include provisions to handle interrupts, computed jumps, the ability to specify more complex trigger conditions, and a "history" function to allow the easy recall of recently used commands.

References

[1] T. N. Mudge, "A Course Sequence in Microprocessor-based Digital Systems Design," <u>IEEE Trans. Education</u>, Vol. E-24, No. 1, Feb. 1981, pp. 14-21.

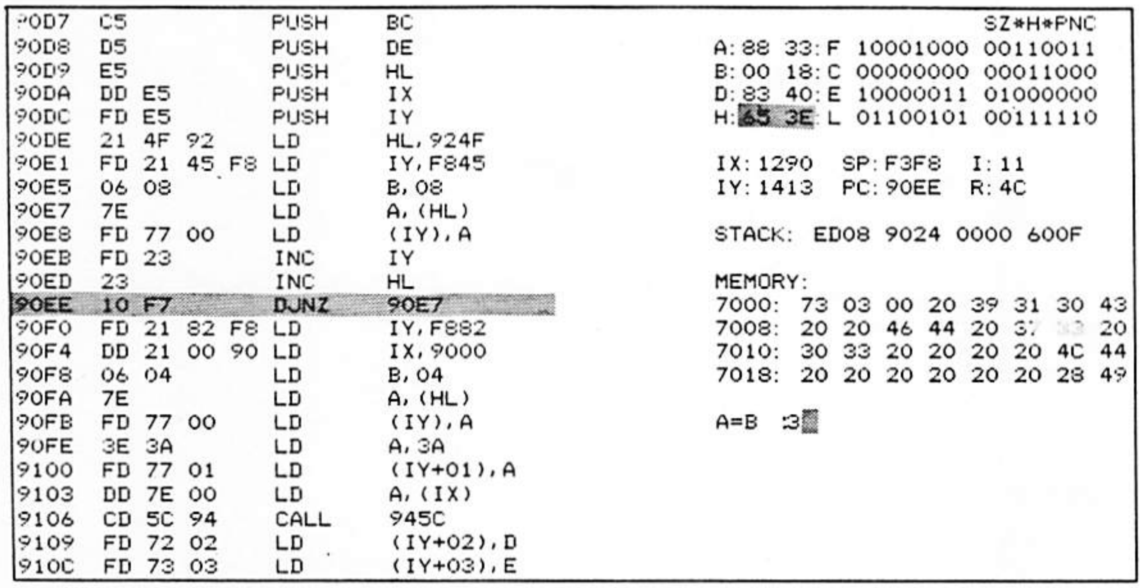
[2] J. B. Baron and D. E. Atkins, "An Educational Laboratory in Contemporary Digital Design," <u>Proc. 2nd Annual Symposium Computer Architecture</u>, Houston, TX, Jan. 1975.

[3] T. N. Mudge, <u>Microprocessor Based Computer and Digital Systems Design Laboratory</u>, Proposal to NSF, Grant No. SER78-13889, Sep. 1978.

[4] <u>Microcomputer Data Book</u>, Mostek, 1979.

YEARLY ENROLLMENT FIGURES								
Data						Estimates		
COURSE	1975	1976	1977	1978	1979	1980	1981	1982
ECE 365	185	192	208	236	245	326	350	350
ECE 366	68	71	70	67	96	114	120	120
ECE 466	49	44	47	48	51	54	60	60

Table 1.



```
90D7  C5           PUSH  BC                                   SZ*H*PNC
90D8  D5           PUSH  DE             A:88  33:F  10001000  00110011
90D9  E5           PUSH  HL             B:00  18:C  00000000  00011000
90DA  DD E5        PUSH  IX             D:83  40:E  10000011  01000000
90DC  FD E5        PUSH  IY             H:65  3E:L  01100101  00111110
90DE  21 4F 92     LD    HL,924F
90E1  FD 21 45 F8  LD    IY,F845        IX:1290   SP:F3F8   I:11
90E5  06 08        LD    B,08           IY:1413   PC:90EE   R:4C
90E7  7E           LD    A,(HL)
90E8  FD 77 00     LD    (IY),A         STACK:  ED08 9024 0000 600F
90EB  FD 23        INC   IY
90ED  23           INC   HL             MEMORY:
90EE  10 F7        DJNZ  90E7           7000:  73 03 00 20 39 31 30 43
90F0  FD 21 82 F8  LD    IY,F882        7008:  20 20 46 44 20 37 33 20
90F4  DD 21 00 90  LD    IX,9000        7010:  30 33 20 20 20 20 4C 44
90F8  06 04        LD    B,04           7018:  20 20 20 20 20 20 28 49
90FA  7E           LD    A,(HL)
90FB  FD 77 00     LD    (IY),A         A=B  :3
90FE  3E 3A        LD    A,3A
9100  FD 77 01     LD    (IY+01),A
9103  DD 7E 00     LD    A,(IX)
9106  CD 5C 94     CALL  945C
9109  FD 72 02     LD    (IY+02),D
910C  FD 73 03     LD    (IY+03),E
```

Figure 1. Screen Layout.

```
<command> ::= GO<hex><hex><hex><hex>¦<modify>¦<display>¦<trigger_condition>¦ON¦OF¦AU¦UA¦OL¦N¦QU

<modify> ::= SE<reg>=<hex><hex>¦SE<double_reg>=<hex><hex><hex><hex>¦SE<memory>=<hex><hex>

<display> ::= DI<memory>

<trigger_condition> ::= <condition><taill>¦<tail>

<condition> ::= <reg><relation><rhs_reg>¦<double_reg><relation><rhs_double>¦
                <memory><relation><rhs_memory>¦F=<bit><bit><bit><bit><bit><bit>
<reg> ::= A¦B¦C¦D¦E¦H¦L
<relation> ::= =¦=¦≤¦≥¦<¦>
<rhs_reg> ::= <reg>¦<hex><hex>¦<memory>
<double_reg> ::= BC¦DE¦HL¦SP¦PC¦IX¦IY
<rhs_double> ::= <double_reg>¦<hex><hex><hex><hex>¦<memory>

<memory> ::= @<hex><hex><hex><hex>
<hex> ::= 0¦1¦2¦3¦4¦...¦F
<taill> ::= return¦:<number>return
<tail> ::= <number>return
<number> ::= <hex>¦<hex><hex>¦<hex><hex><hex>¦<hex><hex><hex><hex>
<bit> ::= 0¦1¦X
```

Figure 2. ZIP's Syntax.