Appendix B

Programming Models

A2.1. Instruction coding in the I8086

Abbreviation		Abbreviation	
reg	register	cond	condition code
reg8	8-bit register	sreg	segment register
reg16	register 16 bits	imm	immediate value
mem	memory address		

Abbreviation	
d	Direction (0 = memory to register, 1 = register to memory)
w	Indicator word (1)/byte (0)
s	Sign (indicator set to 1 for extension from 8 to 16 bits)
mod	Mode indicator: − 00 if r/m = 110: direct addressing, otherwise disp = 0 and there is an indirection. The operand addressing mode must be based, indexed or based-indexed. − 01 indirect access with disp in 8 bits − 10 indirect access with disp in 16 bits − 11 two register instruction (reg for the target and r/m for the source)

Abbrev.		Abbrev.		Abbrev.	
reg	Registers	reg	Registers	sreg	Segments
	– 000 AX AL		– 100 SP AH		– 00 ES
	– 001 CX CL		– 101 BP CH		– 01 CS
	– 010 DX DL		– 110 SI DH		– 10 SS
	– 011 BX BL		– 111 DI BH		– 11 DS

Abbreviation	
r/m	Register/memory (access mode). If the mod field is set to 11, r/m specifies the source register and reg the target. Otherwise, the addressing mode is coded by r/m as follows: – 000 DS:[BX+SI+disp] – 001 DS:[BX+DI+disp] – 010 SS:[BP+SI+disp] – 011 SS:[BP+DI+disp] – 100 DS:[SI+disp] – 101 DS:[DI+disp] – 110 DS:[BP+disp] – 111 DS:[BX+disp]
disp	Operand offset
data	Constants

EXAMPLE A2.1. (**Instruction coding**) –

Instructions	Coding
`mov al,[bx]`	`8A07`
`add ax,bx`	`01D8`
`mov es:[bx+di+a],ax`	`26`
	`89410A`

Hexa	code	d	w	mod	reg	r/m	disp
`8A07`	100010	1	0	00	000	111	
`01D8`	000000	0	1	11	011	000	
`26`	001001	1	0				
`89410A`	100010	0	1	01	000	001	0000 1010

`26` applies to the following instruction `mov` using `es` instead of `ds`.

A2.2. Instruction set of the DLX architecture

A2.2.1. *Operations on floating-point numbers*

`addf`, `addd` `subf`, `subd`	Single and double precision add and subtract
`multf`, `multd` `divf`, `divd`	Same as above for multiply and divide

`cvtf2d`, `cvtf2i`
`cvtd2f`, `cvtd2i`
`cvti2f`, `cvti2d`
	Convert floating-point to double precision or integer
	Convert double precision to floating-point or integer
	Convert integer to floating-point or double precision

EXAMPLE A2.2. (**Operations on floating-point numbers**)

```
movi2fp f1,r2   ; Convert f1:=float(r2)
movi2fp f2,r3   ; Convert f2:=float(r3)
divf f3,f1,f2   ; Floating-point division f3:=f1/f2
movfp2i r1,f3   ; Convert r1:=long(f3)
```

A2.2.2. *Move operations*

`lb`, `lbu`, `sb`	Load byte, load byte unsigned, store byte
`lh`, `lhu`, `sh`	Same as above, but for 16-bit words
`lw`, `sw`	Load/store 32-bit words
`lf`, `sf`, `ld`, `sd`	Load/store single and double precision floating-point numbers
`movi2s`, `movs2i`	Transfer between general registers and control registers
`movf`, `movd`	Transfer between floating-point registers, in single or double precision (register pair)
`movfp2i`, `movi2fp`	Transfer between floating-point registers and integer registers (32 bits)
`lhi`	Load the 16 most significant bits of a register

A2.2.3. *Arithmetic and logic operations*

`add, addi, addu, addui`	Add implicit, immediate, signed and unsigned
`sub, subi, subu, subui`	Same as above, but for subtract
`mult, multu, div, divu`	Multiply and divide signed and unsigned
`and, andi`	Implicit and immediate AND
`or, ori`	Implicit and immediate OR
`xop, xopi`	Implicit and immediate exclusive OR
`sll, srl, slli, srli`	Shift left/right logical, implicit and immediate
`sra, srai`	Shift right arithmetic, implicit and immediate
`s__, s__i`	Sets to 1 or 0 based on `lt`, `gt`, `le`, `ge`, `eq` or `ne` condition

EXAMPLE A2.3.–

```
lhi r1,#-3   ; r1 loaded with FFFD0000(16)
srai r1,#1   ; r1 = FFFE8000(16)
lh r2,(r4)   ; Memory read r2:=MEM(r4)
add r2,r2,#1 ; r2:=r2+1
sw (r4),r2   ; Rewrite into memory
```

A2.2.4. *Branches*

`beqz, bnez`	Conditional branch
`bfpt, bfpf`	Test in the status register of the floating-point processor
`j, jr`	Direct or indirect unconditional branch
`jal, jalr`	Jump and link: direct or indirect unconditional branch with link
`trap`	Transfer to the operating system (vectored address)
`rfe`	Return from exception

EXAMPLE A2.4. (**Conditional branches**)

```
slti r1,r2,#3   ; If r2 < 3 then r1:=1 else r1:=0
beqz r1,next    ; If r1=0 then go to next
```

Direct addressing is written in the format `j tag`, `beqz rk,tag`, etc. The assembler calculates a shift (16 or 26 bits) relative to the following instruction and uses it in the instruction code. In the case of indirect addressing, the content of the register is used as the target address.

Branches with links cause the return address (PC+4) to be stored in register `r31`. The return from the subroutine call is then performed using `jr r31`.