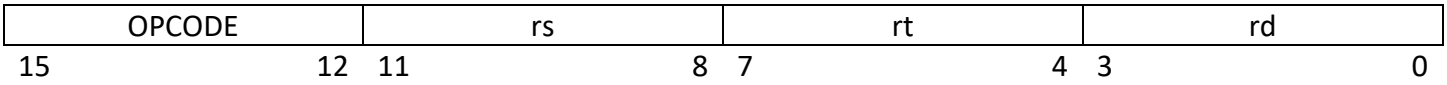
MF16 Instruction Set Architecture User's Guide

General Properties

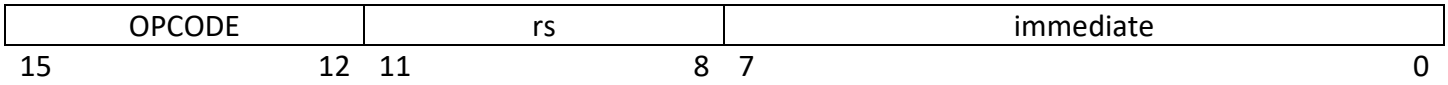
Property	Value
Word size	16 bit
Number of registers	16
Number of Hardware Instructions	16
Number of Pseudo Instructions	14
Number of Instruction types	2
Number of ALU instructions	14
Memory access size	Word
Memory addressing	Byte
Unsigned arithmetic support	None
Floating point arithmetic support	None

Instruction Types

R-type Instructions



I-type Instructions



List of Hardware Instructions

Name	Mnemonic	OPCODE	Type	ALU Involved	Hex
Add	add	0000	R	yes	0
And	and	0001	R	Yes	1
Branch on equal	beq	0010	R	Yes	2
Jump Register	jr	0011	R	Yes	3
Jump register and link	jrl	0100	R	Yes	4
Load upper immediate	lui	0101	I	Yes	5
Load word	lw	0110	R	No	6
Barrel Shift Right (Special feature)	bsr	0111	R	Yes	7
Nor	nor	1000	R	Yes	8
Or	or	1001	R	Yes	9
Or immediate	ori	1010	I	Yes	A
Shift left logical	sll	1011	R	Yes	B
Set on less than	slt	1100	R	Yes	C
Exclusive Or	xor	1101	R	Yes	D
Shift right logical	srl	1110	R	Yes	E
Store word	sw	1111	R	No	F

List of Registers

Name	Number	Description	Preserved across a call?
$0	0	Constant value zero	N/A
$s0	1	Saved temporary	Yes
$s1	2	Saved temporary	Yes
$s2	3	Saved temporary	Yes
$s3	4	Saved temporary	Yes
$t0	5	Temporary	No
$t1	6	Temporary	No
$t2	7	Temporary	No
$t3	8	Temporary	No
$a0	9	Argument	No
$a1	10	Argument	No
$la	11	Label Address	No
$sp	12	Stack pointer	Yes
$rv	13	Return value	Yes
$at	14	Assembler temporary	Yes
$ra	15	Return address	Yes

Instructions

Name	add				**Type**	R		
Syntax	`add rs,rt,rd`							
Description	Add the contents of general purpose register rs to the contents of general purpose register rt and store the 16-bit result in general purpose register rd.							
Operation	GPR[rd] ← GPR[rs] + GPR[rt]							
Encoding	0	0	0	0	rs	rt	rd	
	4				4	4	4	

Name	and				**Type**	R		
Syntax	`and rs,rt,rd`							
Description	Perform bit by bit logical and on the contents of general purpose register rs and the contents of general purpose register rt and store the 16-bit result in general purpose register rd.							
Operation	GPR[rd] ← GPR[rs] & GPR[rt]							
Encoding	0	0	0	1	rs	rt	rd	
	4				4	4	4	

Name	beq				**Type**	R		
Syntax	`beq rs,rt,rd`							
Description	Compare the contents of general purpose register rs and general purpose register rt. If they are equal, branch to the target instruction with a delay of one instruction. The target instruction address is taken from the contents of the general purpose register rd. The low order bit of this address must be zero to keep it word-aligned.							
Operation	T: condition ← (GPR[rs] = GPR[rt]) T+1: if condition then PC ← rd							
Encoding	0	0	1	0	rs	rt	rd	
	4				4	4	4	

Name	jr				**Type**	R		
Syntax	`jr rd`							
Description	Unconditionally jump to the address contained in the general purpose register rd. The contents of general purpose registers rt and rs must be zero. The low order bit of rd must be zero so that the address is word-aligned.							
Operation	T: temp ← GPR[rd] T+1: PC ← temp							
Encoding	0	1	0	0	SBZ	SBZ	rd	
	4				4	4	4	

Name	jrl				Type	R	
Syntax	`jrl rd`						
Description	Unconditionally jump to the address contained in the general purpose register rd. The address of the instruction after the next instruction is placed into the link register $ra. The contents of general purpose registers rt and rs must be zero. The low order bit of rd must be zero so that the address is word-aligned.						
Operation	T: temp ← GPR[rd] $ra ← PC + 1 T+1: PC ← temp						
Encoding	0	1	0	0	SBZ	SBZ	rd
	4				4	4	4

Name	lui				Type	I
Syntax	`lui rs,immediate`					
Description	Place the contents of the 8-bit immediate field into the high order bits of general purpose register rs and set all the low order bits to zero.					
Operation	GPR[rs] ← immediate || 0^8					
Encoding	0	1	0	1	rs	immediate
	4				4	8

Name	lw				Type	R	
Syntax	`lw rs,rt`						
Description	Then load into general purpose register rs, one word from memory at the address stored in general purpose register rt. The lowest order bit of the address must be zero so that it is word-aligned.						
Operation	T: mem ← LoadMemory(WORD,GPR[rt]) T+1: GPR[rt] ← mem						
Encoding	0	1	1	0	rs	rt	SBZ
	4				4	4	4

Name	bsr				Type	R	
Syntax	`bsr rs,rt,rd`						
Description	Right circular shift the contents of general purpose register, inserting lowest order bit into the highest order bit. The shift amount is specified by the lower 4 bits of the contents of general purpose register rs. The result is placed into general purpose register rd.						
Operation	GPR[rd] ← GPR[rt] $\circlearrowright_{shamt}$ (where shamt = $GPR[rs]_{3..0}$)						
Encoding	0	1	1	1	rs	rt	rd
	4				4	4	4

Name	nor				Type	R	
Syntax	`nor rs,rt,rd`						
Description	Perform bit by bit logical or on the contents of general purpose register rs and the contents of general purpose register rt and place the complement of the result in general purpose register rd.						
Operation	GPR[rd] ← ¬(GPR[rs] | GPR[rt])						
Encoding	0	1	1	1	rs	rt	rd
	4				4	4	4

Name	or				Type	R	
Syntax	`or rs,rt,rd`						
Description	Perform bit by bit logical or on the contents of general purpose register rs and the contents of general purpose register rt and place the result in general purpose register rd.						
Operation	GPR[rd] ← GPR[rs] | GPR[rt]						
Encoding	1	0	0	1	rs	rt	rd
	4				4	4	4

Name	ori				Type	I
Syntax	`ori rs,immediate`					
Description	Perform bit by bit logical or on the contents of general purpose register rs and the zero extended contents of the immediate field and store the result by overwriting general purpose register rs.					
Operation	GPR[rd] ← GPR[rs] | (0^8 || immediate)					
Encoding	1	0	1	0	rs	immediate
	4				4	8

Name	sll				Type	R	
Syntax	`sll rs,rt,rd`						
Description	The contents of general purpose register rt is shifted left, inserting zeros into low order bits. The shift amount is specified by the lower 4 bits of the contents of general purpose register rs. The result is placed into general purpose register rd.						
Operation	GPR[rd] ← $GPR[rt]_{16\text{-}shamt..0}$ || 0^{shamt} (where shamt = $GPR[rs]_{3..0}$)						
Encoding	1	0	1	1	rs	rt	rd
	4				4	4	4

Name	slt				Type	R	
Syntax	`slt rs,rt,rd`						
Description	Compare the contents of general purpose register rs with the contents of general purpose register rt. If the former is smaller than the latter, set the general purpose register rd to 1.						
Operation							
Encoding	1	1	0	0	rs	rt	rd
	4				4	4	4

Name	xor				Type	R	
Syntax	`xor rs,rt,rd`						
Description	Perform bit by bit exclusive or on the contents of general purpose register rs and the contents of general purpose register rt and place the result in general purpose register rd.						
Operation	GPR[rd] ← GPR[rs] ⊕ GPR[rt]						
Encoding	1	1	0	1	rs	rt	rd
	4				4	4	4

Name	srl				**Type**	R	
Syntax	`srl rs,rt,rd`						
Description	The contents of general purpose register rt is shifted right, inserting zeros into high order bits. The shift amount is specified by the lower 4 bits of the contents of general purpose register rs. The result is placed into general purpose register rd.						
Operation	GPR[rd] ← 0^{shamt} || $GPR[rt]_{shamt..0}$ (where shamt = $GPR[rs]_{3..0}$)						
Encoding	1	1	1	0	rs	rt	rd
	4				4	4	4

Name	sw				**Type**	R	
Syntax	`sw rs,rt`						
Description	Store in memory, the contents of general purpose register rs at the address specified by the contents of general purpose register rt. The lowest order bit of the address must be zero so that it is word-aligned.						
Operation	T: temp ← GPR[rt] T+1: LoadMemory(WORD,temp)						
Encoding	1	1	1	1	rs	rt	SBZ
	4				4	4	4

List of Pseudo Instructions

Name	Mnemonic
Load immediate	li
Add immediate	addi
And immediate	andi
Subtract	sub
Load word offset	lwo
Store word offset	swo

Name	Mnemonic
Push onto stack	push
Pop from stack	pop
Call a subroutine	call
Jump	jmp
Jump if equal	jeq
sll by immediate	slli

Name	Mnemonic
srl by immediate	srli
bsr by immediate	bsri

Note: The greyed-out instructions were never used in demo programs.

Pseudo-instructions

Name	li	**Number of hardware instructions: 2**
Syntax	`li rs,immediate`	
Description	Place the 16-bit immediate value into general purpose register rs.	
Translated Assembly Code	`lui rs,immediate`$_{15..8}$ `ori rs,immediate`$_{7..0}$	

Name	addi	**Number of hardware instructions: 3**
Syntax	`addi rs,immediate,rd`	
Description	Add the contents of general purpose register rs to the 16-bit immediate value and store the 16-bit result in general purpose register rd.	
Translated Assembly Code	`li $at, immediate` `add rs,$at,rd`	

Name	andi	**Number of hardware instructions: 3**
Syntax	`andi rs,immediate,rd`	
Description	Perform bit by bit logical and on the contents of general purpose register rs and the 16-bit immediate value and store the 16-bit result in general purpose register rd.	
Translated Assembly Code	`li $at, immediate` `and rs,$at,rd`	

Name	sub	**Number of hardware instructions: 13**
Syntax	`sub rs,rt,rd`	
Description	Subtract the contents of general purpose register rt from the contents of general purpose register rs and store the 16-bit result in general purpose register rd. Using the same register as rt and rd is not allowed.	

Translated Assembly Code	`push rt`
	`nor rt, $0, rt`
	`addi rt, 0x1, rt`
	`add rs, rt, rd`
	`pop rt`	

Name	lwo	Number of hardware instructions: 4
Syntax	`lwo rs,rt,immediate`	
Description	Then load into general purpose register rs, one word from memory. The memory address is specified by adding the immediate value to the address stored in general purpose register rt. The lowest order bit of the address must be zero so that it is word-aligned.	
Translated Assembly Code	`addi rt, immediate, $at` `lw rs, $at`	

Name	swo	Number of hardware instructions: 4
Syntax	`swo rs,rt,immediate`	
Description	Store the contents of general purpose register rs into memory. The memory address is specified by adding the immediate value to the address stored in general purpose register rt. The lowest order bit of the address must be zero so that it is word-aligned.	
Translated Assembly Code	`addi rt, immediate, $at` `sw rs, $at`	

Name	push	Number of hardware instructions: 4
Syntax	`push rs`	
Description	Decrement the content of the stack pointer by a word size and store the contents of general purpose register rs on top of the stack.	
Translated Assembly Code	`addi $sp, 0xFFFE, $sp` `sw rs, $sp`	

Name	pop	Number of hardware instructions: 4
Syntax	`pop rs`	
Description	Load a word from top of the stack into general purpose register rs and increment the content of the stack pointer by a word size.	
Translated Assembly Code	`lw rs, $sp` `addi $sp, 0x2, $sp`	

Name	call	Number of hardware instructions: 11
Syntax	`call %func`	
Description	Call a function specified by label %func. The label address is first put into $la register. The contents of $ra register are saved onto the stack. A jump and link is then performed on the address stored in $la. Finally, the $ra is restored from the stack after returning from the function.	

Translated Assembly Code	`li $la, %func`
	`push $ra`
	`jrl $la`
	`pop $ra`	

Name	jmp	Number of hardware instructions: 3
Syntax	`jmp %label`	
Description	Jump to the instruction address specified by label %label. The address of the %label is first stored in $la. An unconditional jump is then performed to the address stored in $la.	
Translated Assembly Code	`li $la, %label` `jr $la`	

Name	jeq	Number of hardware instructions: 3
Syntax	`jeq rs,rt,%label`	
Description	Compare the contents of general purpose register rs and general purpose register rt. If they are equal, branch to the target instruction specified by label %label. The address of the %label is first stored in $la. A conditional jump is then performed to the address stored in $la.	
Translated Assembly Code	`li $la, %label` `beq rs, rt, $la`	

Name	slli	Number of hardware instructions: 3
Syntax	`slli shamt,rt,rd`	
Description	The contents of general purpose register rt is shifted left, inserting zeros into low order bits. The shift amount is specified by the 4 bits of the shamt field. The result is placed into general purpose register rd.	
Translated Assembly Code	`li $at, shamt` `sll $at, rt, rd`	

Name	srli	Number of hardware instructions: 3
Syntax	`srli shamt,rt,rd`	
Description	The contents of general purpose register rt is shifted right, inserting zeros into high order bits. The shift amount is specified by the 4 bits of the shamt field. The result is placed into general purpose register rd.	
Translated Assembly Code	`li $at, shamt` `srl $at, rt, rd`	

Name	bsri	Number of hardware instructions: 3
Syntax	`bsri shamt,rt,rd`	
Description	Right circular shift the contents of general purpose register, inserting lowest order bit into the highest order bit. The shift amount is specified by the 4 bits of the shamt field. The result is placed into general purpose register rd.	
Translated Assembly Code	`li $at, shamt` `bsr $at, rt, rd`	

Assembler Syntax Rules

Comments	'#' symbol indicates the beginning of a line comment. All text after the first occurrence of '#' will be ignored.
Blank lines	Blank lines are ignored
Label definition	Label definitions must be the first token on the line immediately followed by a ':' character. Labels can contain upper and lower case characters and digits. Example: `Label1: add $t1, $0, $t1`
Label usage	Labels can be addressed by using the '%' character immediately followed by the label name. Example: `call %Label1`
Directives	Current directive usage is limited to asciiz and must follow the format below: `Labelname: .asciiz your text goes here` Note that the remainder of the line after the first whitespaces following .asciiz is taken as data. The data is appended with \n\0. Directives can come anywhere within the assembly code.
Registers	All register names must start with '$' character.
Immediate values	Only hexadecimal immediate values are allowed. Hex values must start with "0x".

Control Signals

Signal	Values	Description
writePC	1 0	Enable/Disable writing to PC register
memAddrSrc	pc regfile	Source of memory address for memory operations.
writeMem	1 0	Indicate read/write memory operation
writeIR	1 0	Enable/Disable writing to Instruction register
regFileWriteSrc	alu mem ir nextPC	Specifies the source of the data being written to a register in the register file.
writeReg	rs rt rd ra no	Specifies if a register is being written to or not. If writing, it specified which register it is. It can be one of the 3 register specified in the instruction or $ra.
doBranch	1 0	Specifies if current instruction involves branching.
aluIn1Src	regfile ir	Specifies the source of first argument to the ALU.
aluOpcode	One of 16 opcodes	The opcode value passed to the ALU from the control unit.
memRequest	1 0	CPU tells the memory unit if a memory access was requested
memDone	1 0	Memory unit tells the CPU if a memory access was completed
init_sig	1 0	Specifies if all registers and CPU state must be reset

MF16 CPU Operation

The table below shows the steps of the CPU operation for all of the 16 instructions.

<table>
<tr><th>CPU FSM State</th><th>Clock Cycle</th><th>Clock State</th><th>Actions</th><th>Memory Unit State</th><th>Description</th></tr>
<tr><td>Initial</td><td>1</td><td>Rising edge</td><td>init_sig = 1</td><td rowspan="6">memReady4Next</td><td rowspan="2">This state is only visited once in the beginning of program execution or upon user intervention for resetting the program.</td></tr>
<tr><td>Falling edge</td><td>All registers are latched with their initial values</td></tr>
<tr><td rowspan="12">Fetch</td><td>2</td><td>Rising edge</td><td>If memory is ready:
writePC = 0
doBranch = 0
writeReg = no
memAddrSrc = pc
writeMem = 0
memRequest = 1</td><td rowspan="12">memReady4Next</td></tr>
<tr><td rowspan="2"></td><td>Falling edge</td><td rowspan="10">Memory unit is busy reading next instruction</td></tr>
<tr><td rowspan="2">3</td><td>Rising edge</td></tr>
<tr><td>Falling edge</td><td rowspan="2">memReady4Ctrl</td></tr>
<tr><td rowspan="2">4</td><td>Rising edge</td></tr>
<tr><td>Falling edge</td><td rowspan="3">memReady4Data</td></tr>
<tr><td rowspan="2">5</td><td>Rising edge</td></tr>
<tr><td>Falling edge</td><td>Memory unit output is now valid</td></tr>
<tr><td rowspan="2">Decode</td><td rowspan="2">6</td><td>Rising edge</td><td>If memory is ready:
memRequest = 0
writeIR = 1</td></tr>
<tr><td>Falling edge</td><td>Instruction register output becomes valid</td></tr>
<tr><td rowspan="11">Execute and Memory Access</td><td>7</td><td>Rising edge</td><td>writeIR = 0

add, and, or, nor, slt, sll, srl, xor, bsr
aluIn1Src = reg
ori, lui
aluIn1Src = ir
beq, jr, jrl
aluIn1Src = reg
doBranch =1
lw
memAddrSrc = reg
writeMem = 0
memRequest = 1
sw
memAddrSrc = reg
writeMem = 1
memRequest = 1</td><td rowspan="9">memReady4Next</td><td rowspan="9"></td></tr>
<tr><td rowspan="2"></td><td>Falling edge</td><td rowspan="8">Memory unit is busy working for sw and lw instructions</td></tr>
<tr><td rowspan="2">8</td><td>Rising edge</td></tr>
<tr><td>Falling edge</td><td rowspan="2">memReady4Ctrl</td></tr>
<tr><td rowspan="2">9</td><td>Rising edge</td></tr>
<tr><td>Falling edge</td><td rowspan="3">memReady4Data</td><td rowspan="3">This period only exists when memory access instructions are being executed (sw and lw).</td></tr>
<tr><td rowspan="2">10</td><td>Rising edge</td></tr>
<tr><td>Falling edge</td></tr>
<tr><td>Write Back</td><td>11</td><td>Rising edge</td><td>If memory is ready:
writePC = 1
memRequest = 0

add, and, or, nor, sll, srl, xor, bsr, slt
regFileWriteSrc = alu
writeReg = rd
ori, lui
regFileWriteSrc = alu
writeReg = rs
jrl
regFileWriteSrc = nextPC
writeReg = ra
lw
regFileWriteSrc = mem
writeReg = rs</td><td>memReady4Next</td><td></td></tr>
<tr><td></td><td>Falling edge</td><td>Register file and PC become valid</td><td></td><td></td></tr>
</table>