1. Simple 42 Documentation

The following documents are the current state of our efforts to document the Simple 42 implementation. They are not complete. They describe:

The microinstruction ROM
The data path
The minor cycle timing
The memory interface control
The microcode assembly language
The microcode

Guy Harriman
David May

17.3.82

CONTENTS

1. Overview

The microinstruction ROM contains the microprogram which controls the data path. At the beginning of each minor cycle, the microinstruction address register, MIR, holds the address in the ROM of the microinstruction to be executed. The MIR register is loaded during each minor cycle, unless the cycle is aborted as a result of an abort signal from the interface controller.

2. Microinstruction ROM

The microinstruction ROM contains 122 words, each of 68 bits. The ROM address is taken from the MIR register, and the addressed word is output as control signals.

3. Microinstruction word format

The 68 output signals from the ROM are divided into several fields.

Some of the fields are used to control a number of independent signals, in which case each signal has a microcode signal name. The other fields are decoded to select one of a number of operations, in which case every possible selection has a microcode selector name.

The fields are listed below; those which are decoded are indicated.

Microinstruction word field	Purpose	Decoded
Microword[13:10,8]	X bus source select	
Microword[16,14,9]	Y bus source select	
Microword[48]	X and Y bus connection	
Microword[28:26]	Carry in multiplexor	*
Microword[60:55]	Alu operations	*
Microword[6:0]	Z bus destination select	
Microword[49,44]	OPD register control	
Microword[53:51,43:41]	Shifting control	
Microword[18:17]	A/B stack control	
Microword[25:21]	A/B setting multiplexor	*
Microword[47:46]	C setting from Carryout	
Microword[31:29]	destination phase disable	*
Microword[40:38]	sign control for DIV	
Microword[34:32]	condition select for MIR[1]	*
Microword[37:35]	condition select for MIR[0]	*
Microword[67:61]	loads into MIR	
Microword[54,50,45, 20:19,15,7]	Interface Control Requests	

3.1. X bus source select

Microcode signal name
XfromA
XfromC
XfromW
XfromPW
XfromI

The A, C, W, PW and I registers have signals which cause them to output to the X bus during the source phase.

3.2. Y bus source select

Microcode signal name
YfromB
YfromOPD
YfromMinus15

The B and OPD registers have signals which cause them to output to the Y bus during the source phase. An additional signal, YfromMinus15, causes the value -15 to be output to the Y bus during the source phase, and is used to load the initial value of a loop counter into the OPD register.

3.3. X and Y bus connection

Microcode signal name	Additional signal names
XmergeY	XfromY YfromX

This signal is used to cause the X and Y busses to behave as a single bus during the source phase. It is used to allow the registers which normally output to the X bus to output to the Y bus, and conversely.

3.4. Carry in multiplexor

Microcode selector name	Additional selector names
NotCarryfrom1	NoCarry Borrow
NotCarryfromNotAslave15	CarryfromAslave15
NotCarryfromCslave0	CarryfromNotCslave0
NotCarryfromNotCslave0	CarryfromCslave0
NotCarryfrom0	Carry NoBorrow

A multiplexor selects one of five single bit sources for the carry input to the alu. The additional names are used because the carry input to the alu is inverted.

The CarryfromAslave15 input is used for sign testing in the MUL, DIV and SEX instructions.

The CarryfromNotCslave0 input is used for the SUBC instruction.

The CarryfromCslave0 input is used for the ADDC instruction.

3.5. Alu operations

Microcode selector name	Operation
ZfromXplusY	X + Y + Carry
ZfromXminusY	X - Y - Borrow
ZfromYminusX	Y - X - Borrow
ZfromOnes	-1
ZfromNotCarry	0 - Carry
ZfromZero	0
ZfromX	X
ZfromY	Y
ZfromXxorY	X >< Y
ZfromXxnorY	~ (X >< Y)
ZfromXorY	X \/ Y
ZfromXandY	X /\ Y
ZfromNotX	~X
ZfromNotY	~Y
ZfromMinusY	(-Y) - Borrow
ZfromYminus	Y - Borrow
ZfromYplus	Y + Carry

These selectors are used to select one of the alu operations, the results of which are output to the Z bus for use during the destination phase.

3.6. Z bus destination select

Microcode
signal name

AfromZ
BfromZ
CfromZ
WfromZ
PWfromZ
IfromZ
OPDfromZ

Every data path register has a signal causing it to take input from the Z bus during the destination phase.

3.7. Operand register controls

Microcode
signal name

OPDNot0
IncOPD

The IncOPD signal causes the OPD register to be incremented using the incrementor. The OPD register is input to the incrementor during the source phase; the result is output to the OPD register from the incrementor during the destination phase.

The OPDNot0 signal is set together with the Next signal in the prefix instructions, NFIX and PFIX. The least significant four bits of the OPD register are loaded from the IB register by the Next signal. If the OPDNot0 signal is set the remainder of the OPD register is loaded from the Z bus shifted left four places. If OPDNot0 is not set, the remainder of the OPD register is cleared.

3.8. Shift controls

Microcode
signal name

AShiftin0from1
CShiftin15fromCarryout
Cshiftin15fromArithLogic
ShiftLeftCA
ShiftRightC
ShiftRightA

AShiftin0from0
CShiftin15from0

Both the A and C registers may be shifted one place left or right. The A register may be shifted using the A slave

register, but without use of the X, Y or Z busses; the C register is shifted through the alu using the X and Z busses. The ShiftRightA signal causes the A register to be shifted one place right without use of the X, Y or Z bus, the most significant bit being loaded from the least significant bit of the Z bus.

The ShiftRightC and ShiftRightA signals can be used together to perform a double length shift of the C and A registers. The ShiftRightC signal loads the C register from the Z bus shifted one place right.

The ShiftLeftCA signal is used to perform a double length shift of the C and A registers. It causes the most significant bit of the A slave register to be output to the least significant bit of the X bus together with the C register, shifted left one place. Also, the A register is shifted left one place without use of the busses.

If the AShiftinOfrom1 signal is set when the A register is shifted left, a 1 bit is loaded into the least significant bit of the A register. If the AShiftinOfrom1 signal is not set (represented by AShiftinOfrom0), a 0 bit is loaded into the least significant bit of the A register.

If CShiftin15fromCarryout is set when the C register is shifted right, the most significant bit of the C register is loaded from the alu carry ouput. If CShiftin15fromArithLogic is set when the C register is shifted right, the most significant bit of the C register is loaded from the multiply shift condition logic. If neither signal is set (represented by CShiftin15from0), a 0 bit is loaded into the most significant bit of the C register.

In any shift operation, values are output to the X and Y busses during the source phase, and input from the Z bus and A slave register during the destination phase.

3.9. A and B register stack control

Microcode signal name
AfromB
BfromA

BfromA causes the A register to be copied to the B register, by loading the B register from A slave register during the destination phase. AfromB causes the B register to be copied to the A register, by loading the A register from B slave register during the destination phase. Neither operation requires the use of the X, Y or Z busses.

3.10. A and B register setting controls

Microcode
selector name

AsetfromZeq0
AsetfromGreaterThan

BsetfromZNoteq0
Bsetfrom0verflow
Bsetfrom1
BsetfromQuotSign
Bsetfrom0

These selectors are used to set the A or B register to one of the truth values, TRUE (represented by -1) and FALSE (represented by 0). A multiplexor selects one of seven single bit sources for the truth value; this bit is loaded into all of the bits in the register to be set during the destination phase.

The BsetfromQuotSign selector is used to generate the double length overflow flag in UDIV and DIV.

3.11. C register setting

Microcode
signal name

CsetfromNotCarryout
CsetfromCarryout

These signals cause the C register to be set to one of the truth values, TRUE (represented by -1) and FALSE (represented by 0). The CsetfromCarryout signal causes the C register to be set to TRUE if the alu carry output is set, FALSE otherwise. The CsetfromNotCarryout signal causes the C register to be set to FALSE if the alu carry output is set, TRUE otherwise. The C register is set during the destination phase.

3.12. Destination phase disable

Microcode
selector name

Disablefrom0
DisablefromZlt256
DisablefromNotDivdSign
DisablefromNotQuotSign
DisablefromNotBslave15
DisablefromNotCslave15

In normal operation of the data path, the results of alu operations are input from the Z bus to a selected register.

This input may be conditionally disabled by one of five conditions selected by a multiplexor. Results are conditionally disabled in this way only for MOVE and DIV operations.

The following signals are not disabled during the destination phase, regardless of the selected condition:

Microcode name
MADDRfromZ
ForceFetch
OPDNot0
Next

The DisablefromZlt256 signal is used to detect port addresses for the MOVE instruction.

The other signals are used for signing and unsigning in the DIV instruction.

3.13. Sign controls for DIV

Microcode signal name
SignDivdfromC15
SignQuotfromZ15
DoingDiv

The DIV instruction is performed by converting the divisor and dividend to positive numbers, performing the UDIV instruction, and converting the quotient and remainder to signed numbers.

To execute the DIV instruction, three latches are loaded before performing UDIV. The sign of the dividend is stored in the latch DIVLATCH[0] by the SignDivdfromC15 signal. The sign of the quotient is stored in the latch DIVLATCH[1] by the SignQuotfromZ15 signal. The latch DIVLATCH[2] is set by the DoingDiv signal. The latches are loaded during the destination phase.

3.14. Condition select for MIR[1]

Microcode selector name
Cond1fromROMFeedback1
Cond1fromAslave0
Cond1fromCarryout
Cond1fromMulStep1
Cond1from1
Cond1fromDivStep0
Cond1fromDivStep1
Cond1fromAslave1

The condition multiplexors are used to allow the next microinstruction word to be conditionally selected. These selectors are used to select the source of bit 1 of the MIR register.

The Cond1fromAslave0 and Cond1fromAslave1 selectors are used to determine the operation to be performed in an unsigned multiply step (UMUL step), depending on the least significant bit of the multiplier.

The Cond1fromCarryout selector is used for the DIV, UDIV and MOVE instructions.

The Cond1fromMulStep1 selector is used only for the MUL instruction. The corresponding condition input is generated by the multiply step control logic MULSTEP.

The Cond1from1 selector is not needed.

The Cond1fromDivStep0 and Cond1fromDivStep1 selectors are used only for the UDIV operation. The corresponding condition inputs are generated by the divide step control logic DIVSTEP.

3.15. Condition select for MIR[0]

Microcode selector name
Cond0fromROMFeedback0
Cond0fromDoingDiv
Cond0fromZeq0
Cond0fromMulStep0
Cond0fromOPDeq0

The condition multiplexors are used to allow the next microinstruction word to be conditionally selected. These selectors are used to select the source of bit 0 of the MIR register.

The Cond0fromDoingDiv selector is used after an unsigned division (UDIV) operation, to determine if the unsigned

division was part of a signed division (DIV) operation. The corresponding condition input is the output of the latch DIVLATCH[2], which is set by the DoingDiv signal.

The Cond0fromZeq0 selector is used for the JNZ and SYNC instructions.

The Cond1fromMulStep0 selector is used only for the MUL instruction. The corresponding condition input is generated by the multiply step control logic MULSTEP.

The Cond0fromOPDeq0 selector is used for microcode loop exits in multiply, divide and shift operations.

3.16. Loads to MIR

Microcode labels are used to define the values to be stored in this field.

The MIR register may be loaded from these signals, to provide sequence control.

3.17. Interface control requests

Microcode signal name
MADDRfromZ
DATAINfromZ
Write
Byte
ForceFetch
Next
RunningfromZNoteq0
Read

These signals make requests to the memory interface control. If the interface control is unable to accept a request, it responds with a cycle abort signal.

A data write request is made using MADDRfromZ to provide the address, with Write set. A data read request is made using MADDRfromZ to provide the address, with Write not set (usually represented by Read, which is defined to be 0). The Byte signal may be set for either read or write requests, to perform load and store byte instructions. The XfromDATAIN signal is used to load the data read from memory onto the X bus.

On the last cycle of each instruction, a request for the next instruction is made using the next signal. This may result in an instruction fetch from memory, in which case the MADDR register is loaded from the incrementor output, shifted right one place. The next signal causes the least

significant four bits of the OPD register to be loaded from the IB register, the MIR register to be loaded with the entry address from the pre-decoder, and the I register to be loaded from the incrementor output.

When the I register is loaded, the ForceFetch signal is used, together with MADDRfromZ to cause an instruction fetch from memory.

The RunningfromZNoteq0 signal is used to set the source of instructions, depending on an operand of the SWITCH instruction.

4. MIR register

The MIR register is a seven bit register, allowing up to 128 microinstructions of which 122 are used.

The MIR register may be loaded from the output of the pre-decoder, or from a field in the microinstruction. Also, part of the MIR register may be loaded from condition inputs from the data path.

4.1. Input from the pre-decoder

During the last minor cycle of each instruction, MIR is loaded from the pre-decoder with the entry address of the next instruction, s_EntryAddress[6:0]. The most significant bit of all entry addresses is 0.

4.2. Input from the microinstruction word

During each minor cycle of an instruction except the last, the most significant five bits of MIR, MIR[6:2] are loaded from the field in the microinsruction output word, s_ROMFeedback[6:2]. The two least significant bits, MIR[1] and MIR[0] are loaded from the condition multiplexors CM[1] and CM[0] respectively. The condition multiplexors, CM[1] and CM[0] may take their inputs from s_ROMFeedback[1] and s_ROMFeedback[0] respectively, allowing the whole of MIR to be loaded from s_ROMFeedback[6:0].

5. Condition multiplexors

There are two condition multiplexors, CM[1] and CM[0], which are used to select the source of the least significant bits of MIR, MIR[1] and MIR[0]. Each condition multiplexor is controlled by a three bit field in the microinstruction word, and may therefore select one of eight inputs.

5.1. Condition multiplexor one

Condition multiplexor one, CM[1], is controlled by the three bit field s_M1mux[2:0] in the microinstruction word.

The condition inputs corresponding to the values of s_M1mux[2:0] are:

s_M1mux[2:0]	Microcode selector name	HDL signal name
0	Cond1fromROMfeedback1	s_ROMFeedback[1]
1	Cond1fromAslave0	s_Aslave0
2	Cond1fromCarryout	s_Carryout
3	Cond1fromMulstep1	s_MulStep1
4	Cond1from1	s_VDD
5	Cond1fromDivStep0	s_DivStep0
6	Cond1fromDivStep1	s_DivStep1
7	Cond1fromAslave1	s_Aslave1

The s_Mulstep1 input is generated by the multiply step control logic MULSTEP.

The s_DivStep0 and s_DivStep1 inputs are generated by the divide step control logic DIVSTEP.

5.2. Condition multiplexor zero

Condition multiplexor zero, CM[0], is controlled by the three bit field s_M0mux[2:0] in the microinstruction word.

The condition inputs corresponding to the values of s_M0mux[2:0] are:

s_M0mux[2:0]	Microcode selector name	HDL signal name
0	Cond0fromROMFeedback0	s_ROMFeedback[0]
1	Cond0fromDoingDiv	s_DoingDiv
2	Cond0fromZeq0	s_Zeq0
3	Cond0fromMulStep0	s_MulStep0
4	Cond0fromOPDeq0	s_OPDeq0

The s_DoingDiv input is the output of the latch DIVLATCH[2], which is set by the DoingDiv signal.

The s_Mulstep0 input is generated by the multiply step

control logic MULSTEP.

5.3. Multiply step control logic

After each multiply step, the two least significant bits of the A slave register are used to determine which of three possible step operations is to be performed in the next step. The multiply steps are terminated when the OPD register is zero.

Aslave[1]	Aslave[0]	OPDeq0	Operation
0	0	0	Shift
0	1	0	Add and Shift
1	0	0	Subtract and Shift
1	1	0	Shift
x	x	1	Terminate

There are therefore four possible operations, which are encoded as two condition inputs to the condition multiplexors as follows:

Operation	s_MulStep1	s_Mulstep0
Terminate	0	0
Shift	0	1
Add and Shift	1	0
Subtract and Shift	1	1

The microcode uses these condition inputs to control the next step to be performed.

5.4. Divide step control logic

After each divide step, the alu carry output and the most significant bit of the C slave register are used to determine which of two possible step operations is to be performed in the next step. The divide steps are terminated when the OPD register is zero.

The two possible step operations are:

Add and Shift
Subtract and Shift

After the add and shift operation, the next operation is determined by the s_DivStep0 condition input which is generated as follows:

s_DivStep0 = XOR (s_Carryout, s_Cslave[15])

After the subtract and shift operation, the next operation is determined by the s_DivStep1 condition input which is generated as follows:

```
s_DivStep1 = XOR ( s_NotCarryout, s_Cslave[15] )
```

The s_DivStep0 input is needed because the alu borrow output is the inverted carry output.

CONTENTS

1. Overview

The data path consists of ten single word registers, connected by three busses. An arithmetic logic unit (alu) takes its sources from two of the busses, and outputs its results to the third bus. In addition, certain registers are connected by private paths, and/or have special logic associated with them.

2. Data path

2.1. Registers

The registers fall into two groups; the data path registers and the memory interface registers.

The data path registers are:

A	alu operand register
B	alu operand register
C	auxiliary alu operand register
W	workspace pointer
PW	previous workspace pointer
I	instruction pointer
OPD	instruction operand register

The memory interface registers are:

IB	instruction buffer
DATAIN	buffer for data read from memory
MADDR	buffer for address to memory

2.2. Busses

The three busses are:

X the source of the primary operand to the alu

Y the source of the secondary operand to the alu

Z the result of the alu

2.3. Arithmetic logic unit

The arithmetic logic unit takes single word inputs from the X and Y busses, and outputs a single word result to the Z bus. A carry input and carry output is provided for arithmetic operations. The operation to be performed is selected by a six bit selector field the microinstruction word, s_Aluop[5:0].

The s_Aluop[5] signal enables the alu carry path, and is always set for arithmetic operations. The s_Aluop[4] signal is used to invert the Y input; if s_Aluop[4] is not set, the Y input is inverted.

2.3.1. Logical operations

The alu performs logical operations of the form:

Z := X operation Y

The values which may be output to the Z bus by the alu are:

s_Aluop[5:0]	Microcode selector name	Value
0x0000	ZfromZero	0
000001	ZfromXandY	X /\ Y
000010		X /\ ~Y
0x0011	ZfromX	X
000100		~X /\ ~Y
000101	ZfromXxnorY	~ (X >< Y)
000110	ZfromNotY	~Y
000111		X \/ ~Y
001000		~X /\ Y
001001	ZfromY	Y
001010	ZfromXxorY	X >< Y
001011	ZfromXorY	X \/ Y
0x1100	ZfromNotX	~X
001101		~X \/ Y
001110		~X \/ ~Y
0x1111	ZfromOnes	-1

2.3.2. Arithmetic operations

The alu performs arithmetic operations of the form:

```
Z := operation (X, Y, Carryin)
```

The Carry input is used to provide carry for addition and borrow for subtraction. The carry input signal, s_Notc[0] is the inverted carry input, so:

```
Carryin = NOT s_Notc[0]
```

The values which may be output to the Z bus by the alu are:

s_Aluop[5:0]	Microcode selector name	Purpose	Value
110101	ZfromXminusY	(X-Y) - Borrow	(X + ~Y) + Carryin
100101	ZfromXplusY	(X+Y) + Carry	(X + Y) + Carryin
110110	ZfromMinusY	(0-Y) - Borrow	(~Y) + Carryin
100110	ZfromYplus	Y + Carry	Y + Carryin
101001	ZfromYminus	Y - Carry	Y + ~Carryin
101010	ZfromYminusX	(Y-X) - Borrow	(~X + Y) + Carryin
1x0000	ZfromNotCarry	- Carry	0 + ~Carryin

The other possible arithmetic operations are not particularly useful, and are not used.

2.3.3. Carry in multiplexor

A multiplexor, CARRYINMULTIPLEXOR, controlled by s_Cinmux[2:0] selects one of five single bit sources for the carry input to the alu.

The sources corresponding to the values of Cinmux[2:0] are:

s_Cinmux[2:0]	Microcode selector name	HDL signal name
0	NotCarryfrom1	s_VDD
1	NotCarryfromNotAslave15	s_NotAslave15
2	NotCarryfromCslave0	s_Cslave0
3	NotCarryfromNotCslave0	s_NotCslave0
4	NotCarryfrom0	s_GND

2.4. Incrementor

The incrementor is used to add 1 to the I or OPD register. It takes its input from the I or OPD register; the incremented result may be input to the I or OPD registers, and also to the MADDR register.

2.5. Register to Bus connections

2.5.1. Data path registers

The data path registers fall into two groups; those which output to the X bus and those which output to the Y bus.

The registers which output to the X bus are:

A
C
W
PW
I

The registers which output to the Y bus are:

B
OPD

All of the data path registers take their input from the result bus, Z.

2.5.2. Memory interface registers

The IB register takes its input from the memory interface and outputs to the microcode memory.

The DATAIN register takes its input from the memory interface and outputs to the X bus.

The MADDR register takes its input from the Z bus and outputs to the memory interface.

3. Organisation

3.1. A and B register pair

The A and B registers hold the source operands for most arithmetic and logical operations. The A register may output its contents to the X bus; the B register outputs its contents to the Y bus.

The A register is the destination for most arithmetic and logical operations; however both the A and B registers may take their input from the Z bus.

The A and B registers have associated slave registers, the A and B slave registers.

3.1.1. A and B stack

The slave registers enable the A and B registers to operate as a stack and two paths are provided which allow the A slave register to be copied to B, or the B slave register to A without use of the X, Y or Z bus.

The A register may be loaded from the Z bus whilst the A slave register is copied to the B register. Similarly, the B register may be loaded from the Z bus whilst the B slave register is copied to the A register.

3.1.2. A shifter

The A register may be shifted one position right or left, using the A slave register, without use of the X, Y or Z bus.

3.1.3. A and B setting multiplexor

Either the A or the B register may be set, without use of the X, Y or Z bus, to one of the two truth values, TRUE and FALSE, represented by -1 and 0 respectively. A multiplexor, ABMULTIPLEXOR, controlled by s_ABmux[2:0], selects one of seven single bit sources for the truth value; this bit is loaded into all the bits of the register to be set. If the A register is to be set, the s_SetA signal is set; if the B register is to be set, the s_SetB signal is set.

The sources corresponding to the values of s_ABmux[2:0] are:

s_ABmux[2:0]	Microcode selector name	HDL signal name
0	BSetfrom0	s_GND
1	BSetfromQuotSign	s_QuotSign
2	BSetfrom1	s_VDD

3	ASetfromGreaterThan	s_GreaterThan
4	BSetfromOverflow	s_Overflow
5	BSetfromZNoteq0	s_ZNoteq0
6	ASetfromZeq0	s_Zeq0

3.2. C register

The C register holds the most significant word in double length operations, in which A holds the least significant word. It also holds truth values indicating carry.

The C register takes its input from the Z bus; this input may be direct or shifted right one place.

The C register outputs to the X bus; this output may be direct or shifted left one place.

3.3. W register

The W register takes its input from the Z bus and outputs to the X bus.

3.4. PW register

The PW register takes its input from the Z bus and outputs to the X bus.

3.5. I register

The I register holds a pointer to next instruction to be executed.

The I register inputs from the Z bus, and outputs to the X bus.

The I register also outputs to and inputs from, the incrementor. It may therefore be incremented without use of the alu.

3.6. OPD register

The OPD register holds the operand of the instruction being executed. It is also used to hold temporary values needed during the execution of certain instructions.

The OPD register inputs from the Z bus, and outputs to the Y bus.

The OPD register also outputs to and inputs from, the incrementor. It may therefore be incremented without use of the alu.

The least significant nibble (OPD[3:0]) of the OPD register may take its input from the operand nibble of the next instruction, held in IB. In this case, the remaining nibbles of OPD may be cleared, or taken from the Z bus shifted left four places.

The OPD register may therefore output to the Y bus, and take its input from the shifted Z bus, allowing the operations required for the prefix instructions:

```
OPD := OPD << 4
OPD := (~OPD) << 4
```

The contents of the OPD register are compared with zero, producing the output s_OPDeq0.

3.7. IB register and pre-decoder

The IB register holds the word containing the instruction being decoded. The least significant bit(s) of the instruction register, I, are used to select the instruction within IB. The pre-decoder converts the selected instruction into an entry address in the microcode memory.

The IB register takes its single word input from the memory interface.

Each instruction has a function field, Function[3:0] and an operand field, Operand[3:0]. The IB register outputs the operand field of the selected instruction to the OPD register.

The selected instruction is output from IB to the pre-decoder. The pre-decoder tests the function field of the selected instruction and produces an entry address as follows:

Function nibble	Entry address	
Function[3:0]	EntryAddress[6:4]	EntryAddress[3:0]
0 - D	0	Function[3:0]
E	1	Operand[3:0]
F	2	Operand[3:0]

3.8. DATAIN register and byte router

The DATAIN register is used to hold values of data read from memory.

The DATAIN register takes its input from the memory interface.

The DATAIN register outputs to the X bus via the byte router.

The byte router is used for the load byte and store byte instructions.

3.9. MADDR register

The MADDR register outputs to the memory interface.

The MADDR register takes its input from the Z bus; this input may be direct or shifted right one place. The MADDR register may also input from the incrementor; this input is shifted right one place.

The output of MADDR may be forced to 0, to allow bootstrapping.

The MADDR register is tested to determine if its value lies between 0 and 255, and is therefore the address of a port. This produces the outputs s_MADDR11to8eq0 and s_MADDR15to12eq0.

CONTENTS

1. Overview

Each minor cycle consists of two phases. The source values for operations are output from registers in the source phase; the results of operations are input to registers in the destination phase. The microinstruction control signals fall into three groups; those which are active during the source phase, those which are active during the destination phase, and those which are active throughout the cycle.

2. Data path timing

2.1. Alu operation

In an alu operation, the operands are output to the X and Y busses during the source phase, and the result is taken from the Z bus during the destination phase (unless this is disabled by the destination disable multiplexor). The alu control signals are active throughout the cycle.

2.1.1. Destination disable multiplexor

The destination disable multiplexor, DESTDISABLEMULTIPLEXOR, is controlled by s_dmux[2:0]. In normal operation of the data path, with s_dmux[2:0] set to 0, the results of alu operations are loaded from the Z bus to a selected register during the destination phase. Loading during the destination phase may be conditionally disabled by one of five conditions selected by s_dmux[2:0].

The condition inputs corresponding to the values of s_dmux[2:0] are:

s_dmux[2:0]	Microcode signal name	HDL signal name
0	DisablefromO	s_Gnd
1	DisablefromZlt256	s_Zlt256
2	DisablefromNotDivdSign	s_NotDivdSign
3	DsablefromNotQuotSign	s_NotQuotSign
4	DisablefromnotBslave15	s_NotBslave15
5	DisablefromNotCslave15	s_NotCslave15

2.2. Incrementor operation

In an incrementor operation, the value to be incremented is selected from the I or OPD register during the source phase. The result is taken from the incrementor output during the destination phase.

2.3. A and B register

During every source phase, the A slave register is loaded from the A register and the B slave register is loaded from the B register. The BfromA signal causes the B register to be loaded from the A slave register during the destination phase. Similarly, the AfromB signal causes the A register to be loaded from the B slave register during the destination phase.

The A and B register setting controls are active during the destination phase.

2.4. C register

During every source phase, the C slave register (which contains only bits 0 and 15) is loaded from the C register. The values held in the C slave register are taken during the destination phase, when the C register itself may be loaded from the Z bus.

The C register setting control is active during the destination phase.

3. Microprogram ROM timing

The MIR register is loaded with the address of the microinstruction for the following cycle before the end of the destination phase.

During the last cycle of each instruction, the MIR register is loaded from the pre-decoder during the source phase.

During each minor cycle of an instruction except the last, only the most significant five bits of MIR, MIR[6:2] are loaded during the source phase from the feedback signals, ROMFEEDBACK[6:2]. The least significant two bits of the MIR register, MIR[1:0], are loaded during the destination phase, allowing conditions from the alu to be used to choose the following microinstruction.

4. Memory interface timing

The interface control accepts requests from the interface request signals, and sets the corresponding memory control outputs, during the destination phase. During this destination phase, the address is calculated and loaded into the MADDR register.

The interface control accepts memory control inputs and sets the abort signal during the source phase.

CONTENTS

1. Overview

The interface control contains eight single bit state latches used to record the state of the memory interface. The read and write requests generated by certain microinstructions are combined with the state of these latches to produce the memory interface signals, and the minor cycle abort signal.

2. State latches

2.1. Running

The running latch defines the source of instructions to be executed. If running is false, the source of instructions is port 0. If running is true, the source of instructions is the memory.

The running latch is loaded from an operand of the SWITCH instruction, or from the BOOT pin when RESET is asserted.

2.2. Dbfull

The dbfull latch records the state of the DATAIN register.

The dbfull latch is set false when a read request for data is made to the memory, it is set true when the data arrives.

2.3. Ibempty

The ibempty latch records the state of the IB register.

The ibempty latch is set true when a read request for instructions is made to the memory, it is set false when the instructions arrive. A request for instructions may result from either a forcefetch request, or a next request when only one instruction remains in IB.

2.4. Maddrfull

The maddrfull latch records the state of the MADDR register.

The maddrfull latch is set true when a read request for data or instructions, or a write request, is made to the memory. It is set false when the address is taken by the memory.

2.5. Memenable

The memenable latch records the state of the memory interface. It is true whenever the memory interface is busy.

The memenable latch is set true when a read request for data or instructions, or a write request, is made to the memory. It is set false when data arrives as a result of a read request, or when data is taken by the memory for a write request.

2.6. Write

The write latch records that a write request has been made to the memory.

The write latch is set true when a write request is made to the memory. It is set false when the data is taken by the memory.

2.7. Byte

The byte latch records that a byte request has been made to the memory.

The byte latch is set true when a read byte or write byte request is made to the memory. It is set false when data arrives as a result of a read byte request, or when data is taken by the memory for a write byte request.

2.8. Upperlower

The upperlower latch holds the least significant bit of byte addresses.

The upperlower latch is loaded from the least significant bit of the A register, when the A register is shifted one place right.

3. Abort logic

If a minor cycle cannot proceed because it requires an interface request to be completed, the cycle is aborted. When a cycle is aborted, the results of any operations performed in the cycle are discarded, and the microinstruction register is not loaded with a new state. The conditions for aborting a cycle are described below.

3.1. Instruction fetch

If a next request is made when the instruction buffer is empty (ibempty true), cycles are aborted until the instruction buffer is loaded from the memory.

3.2. Write request

A memory write involves two consecutive cycles, the first to supply the address and the second to supply the data. On the first cycle of a write request, MADDR is loaded (maddrfull set true). The second cycle of a write request is aborted until the address is taken by the memory (maddrfull set false).

3.3. Previous request incomplete

A read or write request may occur before a previous request is completed (maddrfull set true). In this case, the cycle is aborted until the previous request is completed (maddrfull set false).

3.4. Data read request

A memory data read involoves two cycles, the first to supply the address and the second to take the data. These two cycles need not be consecutive. Data from memory is loaded into the DATAIN register (dbfull set true). It is output from the DATAIN register to the X bus on the second cycle of the data read. The second cycle is aborted until the data read request is completed (dbfull set false).

CONTENTS

1. Overview

The microcode assembly language is used to define the contents of the microcode ROM. The microcode assembler translates the assembly language microinstructions into the corresponding bit patterns held in the ROM. A ROM optimiser is also used to re-organise the ROM and eliminate unnecessary cells.

2. Microcode assembly language

2.1. Microprogram

microprogram = WITH **definitions** DO **microinstructions**

2.2. Definitions

definitions = **definition** { **definition** }

definition = **fielddefinition** |
valuedefinition

A definition introduces one or more identifiers for use in subsequent definitions and commands. Each identifier is associated with a bitpatterm, the length of which is the length of a microinstruction.

2.2.1. Field definition

fielddefinition = FIELD Microword **range**
fieldvaluedefs ;

range = HDL range

fieldvaluedefs = **fieldvaluedef** { **fieldvaluedef** }

fieldvaluedef = **identifier** = **bitpattern**

Field definitions are used to introduce one or more identifiers, and associate them with the values which may be taken by a specified field in the microinstruction. The field is specified by the range following =.

The value of an each identifier introduced by a field definition is a bitpattern in which all of the bits outside the range are zero, and the bits inside the range are specified by the bitpattern following =.

2.2.2. Value definition

```
valuedefinition     = SET valuedefs ;

valuedefs           = valuedef { valuedef }

valuedef            = identifier = bitpattern  |
                      identifier = identifier
```

A value definition introduces one or more identifiers and associates each of them with a bitpattern, or with the value associated with another identifier.

2.2.3. Bitpattern

```
bitpattern          = #B bit { bit }

bit                 = 0  |  1
```

2.3. Microinstruction

```
microinstructions   = microinstruction { ; microinstruction }

microinstruction    = identifier : identifers [ conditional ]

identifiers         = identifier { identifier }

conditional         = identifiers [ ( identifiers ) ]
```

Each microinstruction is translated by forming the inclusive or of the bit patterns associated with each of the identifiers following :, together with the value of the conditional, if there is one.

The identifier preceding : is the label of the microinstruction. It is used to define the address in the ROM where the microinstruction is to be stored. It may also be used in the conditional part of other microinstructions.

The value of the conditional part of the microinstruction is the inclusive or of the bit patterns associated with each of the identifiers up to (. These are identifiers associated with conditional fields of the microinstruction, and each is used to select a condition associated with a single microinstruction address bit. The identifiers enclosed by (and) are identifiers used to label other microinstructions, to which transfers are made depending on the values of the selected conditions.

2.4. Comments

Comments are introduced by the character pair //. All subsequent characters up to the end of the line are ignored.

2.5. Use directive

The directive:

```
USE " filename "
```

causes the entire contents of the file specified to replace the use directive. It may occur anywhere in a microprogram.