UNIVERSITY OF JOENSUU
DEPARTMENT OF COMPUTER SCIENCE
Report Series B

MTACSim — a Simulator for the MTAC

Martti Forsell

Report B-1997-1

ACM D.1.3; D.3.2
UDK 681.3.02

MTACSim — a Simulator for the MTAC

Martti J. Forsell

Department of Computer Science, University of Joensuu, PB111, FIN-80101 Joensuu, Finland, email: mforsell@cs.joensuu.fi

June 27, 1997

Abstract

MTAC (MultiThreaded Architecture with Chaining) is a proposal for general purpose parallel processor architecture. In this paper we introduce MTACSim, which simulates the MTAC. MTACSim provides tools for executing and debugging programs written for MTAC processors. It collects also statistical information on program execution for research purposes. MTACSim runs in Macintosh environment and provides a reasonable graphical user interface.

Keywords: VLIW, instruction level parallelism, processor simulators, multithreading

1. The MultiThreaded Architecture with Chaining

The *MultiThreaded Architecture with Chaining* (MTAC) is a superpipelined parallel processor architecture using VLIW-style instruction scheduling policy [Forsell96]. The main parts of a MTAC processor are a arithmetic and logic units (A_0-A_{a-1}), m memory units (M_0-M_{m-1}), operation register (O), process identification register (ID), r general registers (R_0-R_{r-1}) and sequencer (S). Each part consists of a set of registers and processing elements connected to each others in a chain-like manner. Some subparts of chains are also connected to each others (see Figure 1).

The main features of the MTAC are:

- Multithreaded operation with single instruction length threads and a zero-time switch between threads.
- The number of threads can vary from t_{min} to t_{max}.
- Multiple functional units connected as a chain, so that a unit is able to use the results of its predecessors in the chain.
- A very short clock cycle due to a regular structure without long lines that cannot be pipelined.
- VLIW-style instruction scheduling allowing for a complex coding of instructions
- Compilers are easier to construct than with average VLIW machines due to functional unit chaining allowing for the execution of dependent subinstructions within an instruction.

- No need for on-chip (or off-chip) data or instruction caches or branch prediction. Multithreading is used to hide memory, communication network and branch latencies.
- Completely queued operations — no memory reference message handling overhead.
- Single addressing mode
- Explicit write back instructions

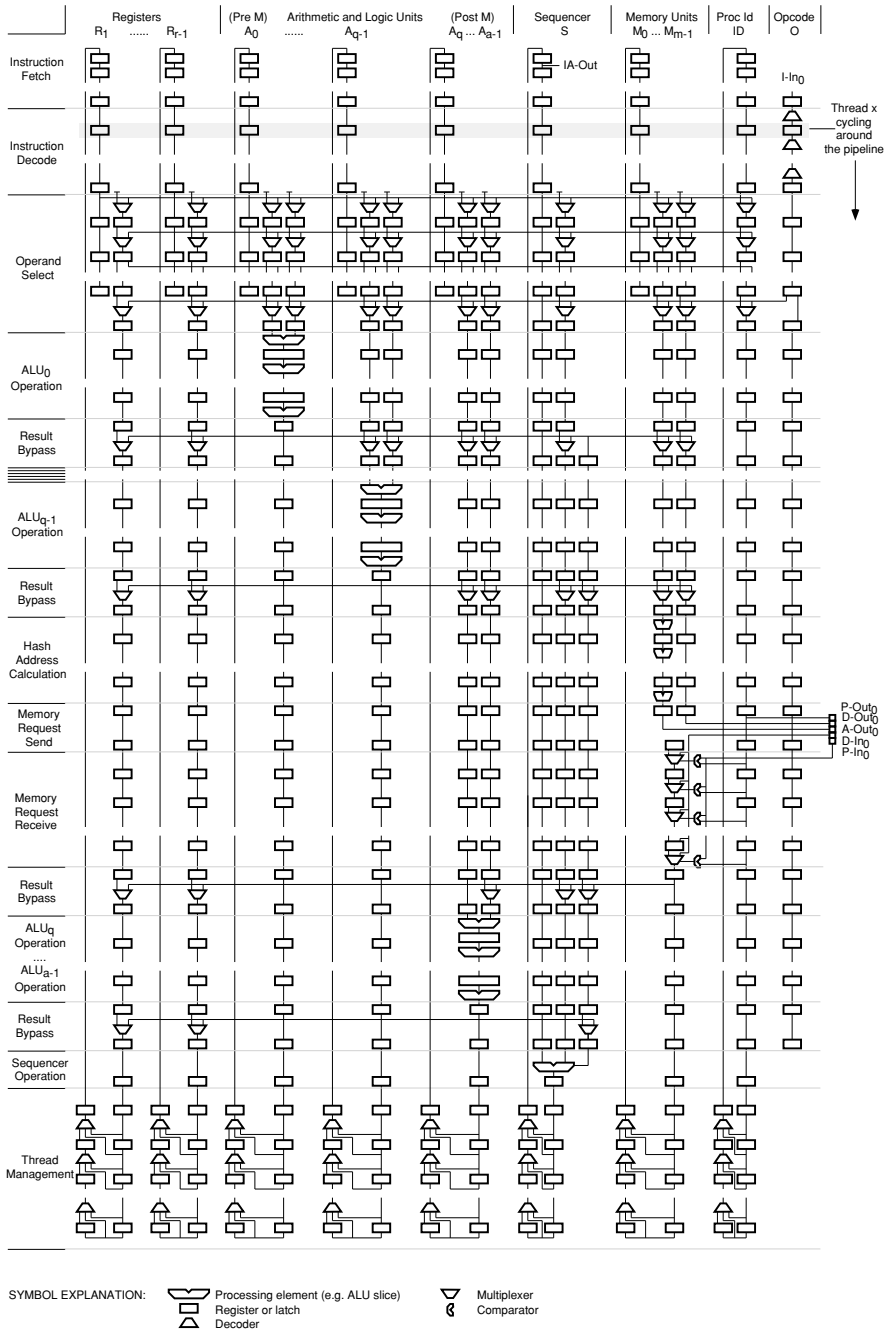

Figure 1 A detailed block diagram of a MTAC processor.

See author's report [Forsell96] for further information on the MTAC.

2. MTACSim

MTACSim is a simulator program for a machine containing multiple MTAC processors connected to a common shared memory. It provides a flexible environment with a reasonable graphical user interface to experiment MTAC programming: The number of processors, the number of threads per processors, and the number of functional units are configurable by the user. MacOS system calls are achieveable through traps and some programming, because MTACSim data memory is embedded into the memory of a host Macintosh.

MTACSim provides tools for executing and debugging MTAC programs as well as tools for collecting statistical information on program execution. In this section we will introduce algorithms, coding conventions, traps and exceptions, and reserved words and error messages of MTACSim.

2.1 Algorithms

The main algorithm of MTACSim is described here shortly. A machine cycle of a MTAC processor is simulated as follows:

For all processors do
- Fetch the next instruction from the instruction memory
- Select operands according to the operation code and send them to appropriate functional units
- Execute pre-memory ALU operations according to operation code
- Execute reads from memory according to operation code
- Execute writes to memory according to operation code
- Execute post-memory ALU operations according to operation code
- Execute operations on the CMP unit according to operation code
- Execute sequencer operation and system call if necessary
- Switch to the next thread

2.2 Coding conventions

Program structure

A MTAC program consists of three segments—the *code segment*, the *data segment* and the *stack and heap segment*. Assigning instructions and data to appropriate segments is described later in this section.

MTACSim places the instructions of a program in the beginning of instruction memory. This area is called code segment. Operation code fetch outside of the code segment causes a bus error exception. Similarly MTACSim places static data defined by storage definition directives in the beginning of data memory. This area is called data segment. The rest of data memory is left for dynamic data placement. It usually contains heap and stack, thus the name of the area is the stack and heap segment.

The starting point of a program must be indicated by _MAIN label in the beginning of code segment. The MEMSIZE symbol contains the address of the last word in data memory. This value can be used to initialize the stack. The THREAD symbol contains the number of threads in a system, i.e., the number of threads per processors times the number of processors in the system.

Source text structure

Each line of a MTAC assembler source text is either a *machine instruction statement* or *directive statement*.

Machine instruction syntax

Machine instruction statements are written in three fields—the *label field*, the *instruction field* and the *comment field*. These fields must be separated by one or more spaces (ASCII code 32) or tabs (ASCII code 9), and must be written in the order given.

The instruction field consists of arbitrary number of instructions which are written in two fields—the *operation field* and the *operand field*.

The label field

The label field is the first field in a source text line. It may be empty or it may contain an identifier. The syntax rules for identifiers are given later in this section.

If the label field contains an identifier, it must begin in the first character position on the line. An identifier in the label field must be terminated by a colon (:), if the label is to be assigned to the data segment. Otherwise it must not terminate with a colon. In this case the label is assigned to the code segment.

The Operation field

The operation field contains the mnemonic operation code, which may be followed by the number of a functional unit. The mnemonic specifies the desired machine instruction. Valid mnemonics are described in appendix A. The assembler ignores uppercase and lowercase distinctions when reading the operation field. The number of the functional unit is required for arithmetic and logical instructions, memory instructions, and write back instructions. It must not be used for compare unit instructions and sequencer instructions.

The Operand field

Many instructions require operands as part of their specification. The operand field follows the operation field and must be separated from it by at least one space or tab. The operand field may be empty, or it may be composed of one or more subfields separated by commas. The operands required for MTAC instructions are described in appendix A.

There are two types of operands—constants and identifiers. Complex expressions are not allowed in this version of MTACSim.

Comments

You can insert a comment into your source text in two ways: as a comment field or as a *comment line*. Comments are ignored by the assembler and may contain any characters except return (ASCII $0D). Comments are intended for your use in documenting your program.

A comment field must begin with a semicolon (;). An entire line may be used for a comment by placing semicolon in the first character position of the line.

Identifiers

Names and labels are *identifiers*. The first character of an identifier must be an uppercase or lowercase letter (A..Z, a..z) or an underscore (_).

Directives

Directives are commands, which control the compilation of the code. There are six directives in MTACSim:

ALIGN	Advance the location counter to the next multiple of a value
ASCII	Place a ASCII string constant in the data segment
DEFAULT	Define a storage area in the data segment, which is filled by ordinal words
RANDOM	Define a storage area in the data segment, which is filled by random words
SPACE	Define a storage area in the data segment
WORD	Place hexadecimal constants in the data segment

MTAC directives follow the general format for assembler statements. You write them in three fields, separated by spaces or tabs:

Label field	***Instruction field***		***Comment field***
	Operation field	***Operand field***	
[*identifier*]	*directive name*	*directive parameters*	[*comments*]

A directive name in operation field begins with period (.).

2.3 Traps and exception handling

Traps

MTACSim uses same trap assignment convention as DLXSim (DLXSim is a simulator program for DLX. It is available as a supplement of [Hennessy90]) or from various anonymous ftp servers (ftp://ftp-acaps.cs.mcgill.ca). In current version only _EXIT (#0) and

_PRINTF (#5) are implemented. See the documentation of DLXSim for more information [Hennessy90].

Exception handling

Exceptions are situations, in which the processor is not able to continue execution normally. MTACSim recognizes four kinds of exceptions—*bus errors*, *resets*, *unimplemented traps* and *halts*.

A bus error occurs when the processor tries to access memory outside the instruction segment, the data segment, or the stack and heap segment. This causes MTACSim to halt the execution of the program. A memory reference outside a segment is not allowed, because the data memory of the simulator is embedded into the memory of Machitosh. A random memory reference might crash the MacOS. A bus error normally indicates that there is a bug in the program.

A reset occurs when user selects Boot item in Process menu. This set the whole simulator to the initial state: Memories are emptied, functional units are initialized and PC is set to zero. The contents of the Command and Output windows are not changed.

An unimplemented trap exception occurs when an unimplemented trap subinstruction is executed. This causes MTACSim to halt execution of the program. Calling a trap, which is defined in DLXSim but not in MTACSim has no effect.

A halt occurs when user selects Halt item in Process menu. This causes MTACSim to halt execution of the program.

2.4 Reserved words and error messages

Reserved words

Some words are reserved in MTAC assembler, because they are used to express internal parts of the processor or some vital constants. Among them are:

_MAIN	Starting point of the MTAC program
MEMSIZE	Address of the last memory location
THREAD	Number of the threads in the machine
A0, A1, ...	ALU outputs
M0, M1, ...	Memory unit outputs
O0, O1, ...	Operands
R0, R1, ...	Registers
PC	Program counter

Please also note that R14 is usually used as a stack pointer, R_{r-2} contains the thread identification number, and R_{r-1} is used as a return address storage when executing a JMPL instruction, where r is the number of registers.

Error messages

Due to limitations of MTACSim and MacOS 7.5 an alert containing one of the following messages can appear:

"You must run on 512Ke or later"
"Application Memory Size is too small"
"Not enough memory to run MTACSim"
"Not enough memory to do Cut"
"Cannot do Cut"
"Cannot do Copy"
"Cannot exceed 32,000 characters with Paste"
"Not enough memory to do Paste"
"Cannot create window"
"Cannot exceed 32,000 characters"
"Cannot do Paste"
"Symboltable Overflow"
"Unknown Command"
"Illegal Operand"

One should press the OK button in the alert and try to avoid the action, which caused the problem.

Due to errors in MTAC source code one of the following messages may appear during the MTAC assembler compilation. Number do not carry any specific information currently:

'Illegal ALU number.'	101
'Illegal memory bus number.'	102
'Illegal memory bus number.'	103
'Illegal register number.'	104
'Illegal operand number.'	105
'Illegal Mnemonic.'	106
'Missing directive.'	107
'End of line ignored.'	108
'Illegal label.'	109
'Illegal label.'	110
'Missing directive.'	111
'End of line ignored.'	112
'SymbolTable overflow'	114
'Unknown operand.'	115
'Unknown instruction class.'	116
'Program too large to fit in instruction memory.'	117

One should fix the MTAC source code according to the message and recompile the code. Note that, only one or two errors are reported at a time.

3. The User Instructions

This section discusses installation, configuration, running and translation issues. Running requirements and a short description of graphical user interface for MTACSim are also given.

Requirements

Requirements for running the MTACSim program are

- Machintosh with M68020, M68030, M68040 processor
- System 7.1 or newer
- 8 MB free RAM
- Large display is strongly recommended

Native PowerPC execution is not currently supported. At least recompilation of the program and rewriting of M32.a file would be needed to run native. The M680LC40 emulation in PowerMacs, however, works fine with MTACSim, because FPU instructions are not implemented in this version.

MTACSim is quite demanding application. Apple PowerMacintosh 7500/100 (100 MHz 601, 32 MB memory) executes 950 MTAC instructions per second.

Distribution

MTACSim is distributed as a high density floppy disk or as a 179kB self-extracting Stuffit Lite document *MTACSim.sea*. When extracted the Stuffit Lite creates a folder containing following 13 files:

ADD.T19	17kB	Example program for T19
ADD.T5	17kB	Example program for T5
DLX.data	17kB	Data file containing information on DLX instructions
MTACSim	213kB	Precompiled MTAC Simulator application
MTACSim.c	17kB	C source file
MTACSim.data	17kB	Data file containing information on MTAC instructions and configuration
MTACSim.data.T19-16-16	17kB	MTAC configuration file for T19 simulation
MTACSim.data.T5-1-1	17kB	MTAC configuration file for T5 simulation
MTACSim.h	26kB	Resource include file
MTACSim.make	17kB	Makefile of the program
MTACSim.p	264kB	Pascal source file
MTACSim.r	34kB	Resource definition source file
MTACSimGlue.a	26kB	M68020-assembler source file

MTACSim was programmed with the Macintosh Programmers Workshop (MPW) programming environment version 3.3.1. The build process (compilation, linking and resource setting) requires 8192kB free memory. Thus, you should set the preferred memory amount of MPW Shell to at least 8192kB.

Installing the program

MTACSim can be installed to a hard disk of a Macintosh by copying the contents of distribution floppy to a folder on the hard disk. The actual MTACSim files are compressed into a self-extracting Stuffit Lite document (MTACSim.sea). The extraction can be started just by double-clicking the icon of MTACSim.sea file.

Configuring the simulator

The configuration of MTACSim is described in MTACSim.data text file, which is read when the program is started. The configuration file contains mnemonics of MTAC instructions, size of data memory and number of functional units.

The contents of MTACSim.data file is (only the numbers marked with italic typeface should be changed):

```
ADD                     2       0
SUB                     2       0
MUL                     2       0
.
.
.
INC                     0       8
OP                      1       9
END
InstructionMemorySize           1024
MemorySize                      65536
NumberOfALUs                    4
NumberOfMemoryBusses            4
NumberOfOperands                4
NumberOfRegisters               33
NumberOfRegisterWindows         8

InstructionMemorySize           1024
MemorySize                      4194304
NumberOfALUs                    1
NumberOfMemoryBusses            1
NumberOfOperands                2
NumberOfPreMemoryALUs           1
NumberOfRegisters               33
NumberOfRegisterWindows         8
NumberOfProcessors              16
NumberOfThreads                 16
TraceMemoryReferences           0
```

The distribution of MTACSim contains two example configuration files—MTACSim.data.T5-1-1 and MTACSim.data.T19-16-16. They represent four and eighteen unit implementations of MTAC with 1 single threaded and sixteen sixteen-threaded processors, respectively.

If the zero on the last line of the configuration file is changed to one, MTACSim generates a memory reference trace file of following format:

Clock cycle field
Read/Write field *(R= Read, W = Write)*
Processor number field
Thread number field
Address field
Data field *(Read data not shown)*
Opcode field

2	R	0	0	16287828	0	64
3	W	0	0	16291924	431	67

The graphical user interface

The graphical user interface of MTACSim consists of a menu-bar, seven windows, some dialog boxes and alerts.

The structure of the menu-bar is:

	File	**Edit**	**Process**	**Breakpoint**	**Windows**
About MTACSim...	New	*Undo*	Boot	Insert	Commands
	Open	----------	----------	Remove	Listing
	----------	Cut	Run	----------	Dump
	Close	Copy	Go	Clear All	Output
	Save	Paste	Pass		Processor
	Save As...	Clear	Step		Setup
	Revert		Step P		Statistics
	----------		Step T		
	Import		Halt		
	Page Setup...				
	Print				

	Quit				

Items marked with *italic typeface* are not in use in this version. Most menu items also have keyboard shortcuts.

The state of the processor, the contents of memories as well as commands, output messages and statistics are all presented in seven windows. The windows are:

Commands	Shows the commands that are given to MTACSim
Memory	Shows the contents of the Data Memory
Listing	Shows the contents of the Instruction Memory in symbolic form
Output	Shows the output of simulator and possible system calls
Processor	Shows the state of the processor
Statistics	Shows statistical information on execution of the program
Setup	Not currently used

File references are made with standard MacOS dialog boxes that allow you to browse and select files. Some MacOS-related error messages are notified by alerts. The list of those error messages is shown in Section 2.2.

Managing windows

The windows are typical Macintosh environment windows with go-away region, drag region, grow region, full size region and two scrollbars.

They are directly connected to standard TextEdit managers with the exception of Memory window. You can write and delete text as well as copy and paste like in every normal MacOS application. The size of text is limited to 32000 bytes. This may cause problems when dealing with large programs or long simulations. You may wish to delete a part or the entire content of a window if you run out of space.

Executing MTAC programs

Loading a MTAC program into MTACSim:

> *Select Open item in File menu*
> *Select the file, which should be loaded into MTACSim using the standard file open dialog*

MTACSim will now compile the program to machine language of MTAC according to configuration used. If there are errors in the program error messages are shown in Output window, otherwise the compiled program will now be shown in Listing window in symbolic form.

Running a MTAC program:

> *Select Run item in Process menu*

The program is executed until the processor is reset, the execution is halted, an _EXIT trap is met, a breakpoint is met or an exception occurs. At the same time statistical information on program execution is collected.

Halting the execution of a program:

> *Select Halt item in Process menu*

Continuing suspended execution:

> *Select Go item in Process menu*

Executing a MTAC program stepwise:

> *Select Step item in Process menu*

Only the next instruction of the program is executed. This command will update the statistics.

> *Select Step P item in Process menu*

Instructions are executed until a thread belonging to the same processor becomes into execution. This command will update the statistics.

Select Step T item in Process menu

Instructions are executed until an instruction belonging to the same thread becomes into execution. This command will update the statistics.

Resetting the processor:

Select Boot item in Process menu

Executing a program without collecting statistical information:

Select Pass item in Process menu

Passing can be applied to a block of code by setting breakpoints into the beginning and the end of the block. The execution before the block should be done normally. The block should be executed with Pass command and the rest of the code should be executed normally with Go command.

Setting and removing breakpoints

One can insert up to 10 breakpoints into MTAC code. The execution of a MTAC program is halted if a breakpoint is met. This simplifies debugging and measuring of MTAC programs. The breakpoints are related with the instructions of a program, so each thread will halt if a breakpoint is met.

Inserting a breakpoint into a MTAC code:

Select the row in which the breakpoint should be inserted in Listing window
Select Insert item in Breakpoint menu

Symbol -> should now appear between address and instructions indicating the presence of a breakpoint.

Removing a breakpoint from a MTAC code:

Select the row from which the breakpoint should be removed in Listing window
Select Remove item in Breakpoint menu

Symbol -> indicating the presence of the breakpoint should now disappear.

All breakpoints could be removed by selecting Clear All item in Breakpoint menu.

Collecting statistical information

MTACSim collects statistical information on the execution of the programs. The information is shown on statistics window.

The information is shown according to following format:

MTACSim - A MULTITHREADED VLIW PROCESSOR SIMULATOR

© Copyright Martti Forsell, 1997
Department of Computer Science, University of Joensuu

STATISTICAL INFORMATION ON PROGRAM EXECUTION

PROGRAM:
Name of the Program : max.T19

MACHINE SETUP:
Number of Processors : 16
Number of Threads : 16
Number of ALUs : 12
Number of Memory Busses : 4
Number of Operands : 12
Number of Registers : 32
Start of Data Memory : 0x00F0A874
Size of Data Segment : 65536
Size of Program Memory : 6

GENERAL INFORMATION:
Clock Cycles : 401

INSTRUCTION CLASS FREQUENCIES:

tALUI	*42240*
tAICI	*25344*
tLDI	*19456*
tSTI	*1280*
tBRAI	*4864*
tJMPI	*1*
tREGI	*31232*
tWINI	*0*
tINTI	*1536*
tOPI	*23808*
tNOPI	*20497*

FUNCTIONAL UNIT UTILIZATION:

*U*1000*	*12910*

INSTRUCTION FREQUENCIES:

ADD	*19968*	
SHR	*1024*	
SHL	*768*	
NOP	*20497*	
SLT	*3840*	
SGT	*20480*	
SGE	*1024*	
LD	*19456*	
ST	*1280*	
BNEZ	*4864*	*4352*
TRAP	*1*	*0*

CMD 1536
SEL 20480
WB 31232
OP 23808

OTHER INFORMATION:
Number of operations : 93185
Theoretical minimum : 5226

Statistical information produced by MTACSim can be transferred to worksheet programs like Microsoft Excel for further processing and drawing diagrams by copying the content of statistics window to clipboard and pasting it to the worksheet.

Bugs and limitations

Being a research project resulting to 11000 lines of program code MTACSim has several limitations and bugs.

- Internal data structures are static
- Instruction memory is small due to unefficient coding of instruction words
- The content of a window is limited to 32000 characters
- Floating point instructions are not implemented
- Reconfiguration must be done by hand editing a text file and rerunning the program
- Only two traps are actually implemented
- Assemblers have poor error recovery process

References

[Forsell96] M. Forsell, MTAC—A Multithreaded VLIW Architecture for PRAM Simulation, *Report A-1996-7*, Department of Computer Science, University of Joensuu, Joensuu, 1996.

[Hennessy90] J. L. Hennessy, D. A. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publishers Inc., San Mateo, 1990.

Appendices

A. The complete instruction set of the MTAC

The symbols used for describing MPA operations are

Ax	Result of ALU x
AAx	Input register 1 of ALU x
ABx	Input register 2 of ALU x
AIA	Input register 1 of ALU for integer condition codes
AIB	Input register 2 of ALU for integer condition codes
AIC	Result of ALU for integer condition codes
d	32-bit immediate value
I[x]	Location x of instruction memory
ID	Thread identification register
Mx	Result of memory x operation
Mx[y]	Location y of memory x
MAx	Address register for memory x
MDx	Data register for memory x
MuxY	Multiplexer for Y.
O	Operation register
Ox	Operand x of an instruction
OO	Result of operation code fetch
O1	Immediate operand field of operation register
PC	Program counter
RA	Return address
Rx	Contents of register x (R14 = SP, R30 = ID, and R31 = RA)
Xx	Ax, Mx, Ox or Rx

The instructions of the MTAC are

Load		
LDBn	Xx	Load byte from memory n address Xx
LDBUn	Xx	Load byte from memory n address Xx unsigned
LDHn	Xx	Load halfword from memory n address Xx
LDHUn	Xx	Load halfword from memory n address Xx unsigned
LDn	Xx	Load word from memory n address Xx
Store		
STBn	Xx,Xy	Store byte Xx to memory n address Xy
STHn	Xx,Xy	Store halfword Xx to memory n address Xy
STn	Xx,Xy	Store word Xx to memory n address Xy
Write Back		
WBn	Xx	Write Xx to register Rn. Old value of Rn is preserved if x is r-1.
Operand Input		
OPn	d	Input value d into operand n
Align		
ALBn	Xx,Xy	Align load byte Xx by Xy in ALU n
ALBUn	Xx,Xy	Align load byte Xx by Xy in ALU n unsigned

ALHn	Xx,Xy	Align load halfword Xx by Xy in ALU n
ALHUn	Xx,Xy	Align load halfword Xx by Xy in ALU n unsigned
ASSBn	Xx,Xy	Align store source byte Xx by Xy in ALU n
ASSHn	Xx,Xy	Align store source halfword Xx by Xy in ALU n
ASDBn	Xx,Xy	Align store destination byte Xx by Xy in ALU n
ASDHn	Xx,Xy	Align store destination halfword Xx by Xy in ALU n

Arithmetic

ADDn	Xx,Xy	Add Xx and Xy in ALU n
SUBn	Xx,Xy	Subtract Xy from Xx in ALU n
MULn	Xx,Xy	Multiply Xx by Xy in ALU n
MULUn	Xx,Xy	Multiply Xx by Xy in ALU n unsigned
DIVn	Xx,Xy	Divide Xx by Xy in ALU n
DIVUn	Xx,Xy	Divide Xx by Xy in ALU n unsigned
MODn	Xx,Xy	Determine Xx modulo Xy in ALU n
MODUn	Xx,Xy	Determine Xx modulo Xy in ALU n unsigned
SELn	Xx,Xy	Select Xx or Xy according to the result of previous compare operation in functional unit chain

Logical

ANDn	Xx,Xy	And of Xx and Xy in ALU n
ORn	Xx,Xy	Or of Xx and Xy in ALU n
XORn	Xx,Xy	Exclusive or of Xx and Xy in ALU n
ANDNn	Xx,Xy	And not of Xx and Xy in ALU n
ORNn	Xx,Xy	Or not of Xx and Xy in ALU n
XNORn	Xx,Xy	Exclusive nor of Xx and Xy in ALU n
SHRn	Xx,Xy	Shift right Xx by Xy in ALU n
SHLn	Xx,Xy	Shift left Xx by Xy in ALU n
SHRAn	Xx,Xy	Shift right Xx by Xy in ALU n arithmetic
NOPn		No operation in ALU n
RORn	Xx,Xy	Rotate right Xx by Xy in ALU n
ROLn	Xx,Xy	Rotate left Xx by Xy in ALU n

Compare

SEQ	Xx,Xy	Set IC if Xx equals Xy
SNE	Xx,Xy	Set IC if Xx not equals Xy
SLT	Xx,Xy	Set IC if Xx is less than Xy
SLE	Xx,Xy	Set IC if Xx is less than or equals Xy
SGT	Xx,Xy	Set IC if Xx is greater than Xy
SGE	Xx,Xy	Set IC if Xx is greater than or equals Xy
SLTU	Xx,Xy	Set IC if Xx is less than Xy unsigned
SLEU	Xx,Xy	Set IC if Xx is less than or equals Xy unsigned
SGTU	Xx,Xy	Set IC if Xx is greater than Xy unsigned
SGEU	Xx,Xy	Set IC if Xx is greater than or equals Xy unsigned

ADDCC	*Xx,Xy*	*Add Xx and Xy and set CC*
SUBCC	*Xx,Xy*	*Subtract Xy from Xx and set CC*
MULCC	*Xx,Xy*	*Multiply Xx by Xy and set CC*
MULUCC	*Xx,Xy*	*Multiply Xx by Xy unsigned and set CC*
DIVCC	*Xx,Xy*	*Divide Xx by Xy and set CC*
DIVUCC	*Xx,Xy*	*Divide Xx by Xy unsigned and set CC*

MODCC	*Xx,Xy*	*Determine Xx modulo Xy and set CC*
MODUCC	*Xx,Xy*	*Determine Xx modulo Xy unsigned and set CC*
ADDXn	*Xx,Xy*	*Add with carry Xx and Xy in ALU n*
SUBXn	*Xx,Xy*	*Subtract with carry Xy from Xx in ALU n*
ADDXCC	*Xx,Xy*	*Add with carry Xx and Xy and set CC*
SUBXCC	*Xx,Xy*	*Subtract with carry Xy from Xx and set CC*
TADDn	*Xx,Xy*	*Tagged add Xx and Xy in ALU n*
TSUBn	*Xx,Xy*	*Tagged subtract Xy from Xx in ALU n*
TADDCC	*Xx,Xy*	*Tagged add Xx and Xy and set CC*
TSUBCC	*Xx,Xy*	*Tagged subtract Xy from Xx and set CC*
ANDCC	*Xx,Xy*	*And of Xx and Xy in ALU n and set CC*
ORCC	*Xx,Xy*	*Or of Xx and Xy in ALU n and set CC*
XORCC	*Xx,Xy*	*Exclusive or of Xx and Xy in ALU n and set CC*
ANDNCC	*Xx,Xy*	*And not of Xx and Xy in ALU n and set CC*
ORNCC	*Xx,Xy*	*Or not of Xx and Xy in ALU n and set CC*
XNORCC	*Xx,Xy*	*Exclusive nor of Xx and Xy in ALU n and set CC*

Control

BEQZ	Ox[,Xx]	Branch to Ox if IC equals zero. Otherwise the execution continues from the next instruction or from Xx if supplied
BNEZ	Ox[,Xx]	Branch to Ox if IC not equals zero. Otherwise the execution continues from the next instruction or from Xx if supplied.
FBT	Ox[,Xx]	Floating Branch to Ox if FC equals zero. Otherwise the execution continues from the next instruction or from Xx if supplied.
FBF	Ox[,Xx]	Floating Branch to Ox if FC not equals zero. Otherwise the execution continues from the next instruction or from Xx if supplied.
JMP	Xx	Jump to Xx
JMPL	Xx	Jump and link PC+1 to register RA
TRAP	Xx	Trap
BA	Ox	Branch always to Ox
BN	Ox	Branch newer to Ox
BNE	Ox[,Xx]	Branch on not equal to Ox (BNZ). Otherwise the execution continues from the next instruction or from Xx if supplied.
BE	Ox[,Xx]	Branch on equal to Ox (BZ). Otherwise the execution continues from the next instruction or from Xx if supplied.
BG	Ox[,Xx]	Branch on greater to Ox. Otherwise the execution continues from the next instruction or from Xx if supplied.
BLE	Ox[,Xx]	Branch on less than or equal to Ox. Otherwise the execution continues from the next instruction or from Xx if supplied.
BGE	Ox[,Xx]	Branch on greater than or equal to Ox. Otherwise the execution continues from the next instruction or from Xx if supplied.
BL	Ox[,Xx]	Branch on less than to Ox. Otherwise the execution continues from the next instruction or from Xx if supplied.
BGU	Ox[,Xx]	Branch on greater to Ox unsigned. Otherwise the execution continues from the next instruction or from Xx if supplied.

BLEU	Ox[,Xx]	Branch on less than or equal to Ox unsigned. Otherwise the execution continues from the next instruction or from Xx if supplied.
BCC	Ox[,Xx]	Branch on carry clear to Ox (BGEU). Otherwise the execution continues from the next instruction or from Xx if supplied.
BCS	Ox[,Xx]	Branch on carry set to Ox (BLU). Otherwise the execution continues from the next instruction or from Xx if supplied.
BPOS	Ox[,Xx]	Branch on positive to Ox. Otherwise the execution continues from the next instruction or from Xx if supplied.
BNEG	Ox[,Xx]	Branch on negative to Ox. Otherwise the execution continues from the next instruction or from Xx if supplied.
BVC	Ox[,Xx]	Branch on overflow clear to Ox. Otherwise the execution continues from the next instruction or from Xx if supplied.
BVS	Ox[,Xx]	Branch on overflow set to Ox. Otherwise the execution continues from the next instruction or from Xx if supplied.

Register window

SAVE	*Save register window*
REST	*Restore register window*

Floating Point

FADDn	Xx,Xy	Floating add Xx and Xy in ALU n
FSUBn	Xx,Xy	Floating subtract Xy from Xx in ALU n
FMULn	Xx,Xy	Floating multiply Xx by Xy in ALU n
FDIVn	Xx,Xy	Floating divide Xy by Xx in ALU n
FMODn	Xx,Xy	Floating Xx modulo Xy in ALU n
FSEQ	Xx,Xy	Floating set FC if Xx equals Xy
FSNE	Xx,Xy	Floating set FC if Xx not equals Xy
FSLT	Xx,Xy	Floating set FC if Xx is less than Xy
FSLE	Xx,Xy	Floating set FC if Xx is less than or equals Xy
FSGT	Xx,Xy	Floating set FC if Xx is greater than Xy
FSGE	Xx,Xy	Floating set FC if Xx is greater than or equals Xy
DADDn	Xx,Xy	Double add Xx and Xy in ALU n
DSUBn	Xx,Xy	Double subtract Xy from Xx in ALU n
DMULn	Xx,Xy	Double multiply Xx by Xy in ALU n
DDIVn	Xx,Xy	Double divide Xy by Xx in ALU n
DMODn	Xx,Xy	Double Xx modulo Xy in ALU n
DSEQ	Xx,Xy	Double set FC if Xx equals Xy
DSNE	Xx,Xy	Double set FC if Xx not equals Xy
DSLT	Xx,Xy	Double set FC if Xx is less than Xy
DSLE	Xx,Xy	Double set FC if Xx is less than or equals Xy
DSGT	Xx,Xy	Double set FC if Xx is greater than Xy
DSGE	Xx,Xy	Double set FC if Xx is greater than or equals Xy
CVTFIn	Xx	Convert floating Xx to integer in ALU n
CVTFDn	Xx	Convert floating Xx to double in ALU n
CVTDIn	Xx	Convert double Xx to integer in ALU n
CVTDFn	Xx	Convert double Xx to floating in ALU n
CVTIFn	Xx	Convert integer Xx to floating in ALU n
CVTIDn	Xx	Convert integer Xx to double in ALU n