esa
estec

european space research
and technology centre

Ref: WSD/JG/160/NL
Issue 1, 27-11-1995

ERC32 instruction timing

Prepared by J.Gaisler
Spacecraft Control and Data System Division
Automation and Informatics Department
ESA/ESTEC

Keplerlan 1 - Noordwijk - Netherlands
Mail address: Postbus 299 - 2200 AG - Noordwijk - The Netherlands
Tel: 071-5656565 - Telex: 39098 - Fax: 071-5656040

1 Introduction

1.1 Scope

This document describes the instruction timing for the 90C601RT and 90C602RT in an ERC32 system.

1.2 Applicable documents

AD1 IU-RT specification, issue 6, AMS/IURT/0017/CLG, Matra MHS 1995

AD2 FPU-RT specification, issue 6, AMS/FPURT/0018/VS, Matra MHS 1995

AD3 MEC device specification, issue 7, MCD/SPC/0005/SE, Saab Ericsson Space 1995

2 Instruction timing

Integer instructions consist of up to four cycle types; fetch, load, store and internal cycles. Table 1 indicates the cycle decomposition for 90C601RT instructions.

Instruction	Internal cycles	Fetch cycles	Load cycles	Store cycles
LD	0	1	1	0
LDD	0	1	2	0
ST	1	1	0	1
STD	1	1	0	2
JMPL, RETT	1	1	0	0
LDST, SWAP	1	1	1	1
All other instructions	0	1	0	0

Table 1: 90C601RT instruction timing

Data dependencies can add one internal cycle to the instruction timing. This occurs if an instruction uses data loaded in the previous cycle, if the delay slot instruction of a CALL uses r[15], or if the delay slot instruction of a JMPL uses the destination register of JMPL. The extra cycles occur relatively rarely, most compilers know about the dependencies and try to schedule around them.

The instruction timing of the floating point instructions executed by 90C602RT is somewhat more difficult to calculate. The timing depends on both the data used in the calculations, and resource dependencies between consecutive FP instructions. Since the FPU can execute one FP instruction in parallel with the IU, the effective timing depends on whether a new FP instruction arrives before the previous is completes. If so, the FPU will halt the system until the first FP instruction completes. Table 2 below gives the FP instruction timing as indicates in the 90C602RT data sheet. This timing indicates the non-parallel mode, i.e. how many cycles the FP instruction needs to complete inside the FPU. To launch an FP instruction only takes one instruction fetch cycle - typically one clock cycle in an ERC32 system. The effective CPI for FP instruction depends largely on how well the compiler schedules the instructions, avoiding data dependencies and maximizing concurrent operation with the IU. The worst case timing occurs only when denormalized numbers are used as source for operations.

Floating-point load and store instructions are executed partly by the IU, and are not mentioned in table 2. They execute in the same number of clock cycles as the corresponding integer instruction. The FP load/store instructions can be executed in parallel with any ongoing FP operation as long as no resource dependency exists. Table 3 indicates when a dependency would occur and the FP load/store instruction would have to wait until the current FP instruction finishes.

Instruction	best case	Typical case	Worst case
FABS	2	2	2
FADDS	4	4	17
FADDD	4	4	17
FCMPS	4	4	15
FCMPD	4	4	15
FCMPES	4	4	15
FCMPED	4	4	15
FDIVS	6	20	38
FDIVD	6	35	56
FMOVS	2	2	2
FMULS	5	5	25
FMULD	7	9	32
FNEGS	2	2	2
FSQRTS	6	37	51
FSQRTD	6	65	80
FSUBS	2	4	17
FSUBD	4	4	17
FDTOI	7	7	14
FDTOS	3	3	16
FITOS	5	6	13
FITOD	4	6	13
FSTOI	6	6	13
FSTOD	2	2	14

Table 2: 90C602R T instruction timing

Instruction	Dependency
LDF, LDDF	A load instruction must not overwrite the source or destination of any FPop that has not completed execution.
STF, STDF	A store operation may not access an FP register that is the destination register of a FPop that has not yet finished execution.
LDFSR, STFSR	Any FP instruction must finish before a LDFSR/STFSR may start. No FP instruction can start before a LDFSR/STFSR is finished.
STDFQ	If a STDFQ is issued when the FP queue is empty , any on-going FP instructions are halted until the STDFQ completes.

Table 3: 90C602R T dependencies

3 ERC32 performance calculation

To calculate the instruction timing in an ERC32 system, a mapping between processor cycles and system clocks must be done. Table 4 indicates the number of system clocks for each type of cycle. The number of waitstates is denoted as "n".

Cycle type	System clocks
IU internal cycle	1
RAM fetch, load & store	1 + n
boot-PROM fetch, load & store	1 + 4n
I/O area load and store	2 + n
Exchange memory load & store	2 + n

Table 4: ERC32 load & store cycle timing

In an ERC32 system, the store byte and half-word instructions (STH and STB) are converted to a read-modify-write cycle in order to update the EDAC check-bits properly. Table 5 shows the cycle composition for these instructions.

Instruction	Internal cycles	Load cycles	Store cycles
STB, STH	3	1	1

Table 5: STB, STH timing in an ERC32 system

A typical ERC32 system executes applications from zero-waitstate RAM, and has zero or one waitstate during write operations. In table 6, the instruction timing for three cases are shown; zero waitstate load and store, one waitstate during store and boot-PROM access with 2 waitstates.

Instruction	0 ws	1 ws (write)	boot-PROM
LD	2	2	13
LDD	3	3	25
ST	3	4	14
STD	4	6	26
STB, STH	5	6	-
JMPL, RETT	2	2	2
All other instructions	1	1	13

Table 6: Instruction timing in an ERC32 system

Simulations have shown that the effective CPI for integer instructions in zero-waitstate systems is 1.2 - 1.4, while one waitstate during store cycles increases the CPI to 1.3 - 1.5. Programs compiled with gcc-2.7 and gnat-2.0.7 has shown that the effective CPI for single precision FP instructions is about 2.5 and for double precision 4.0. The CPI for FP instruction is mostly unaffected by waitstates since FP operations run in parallel with IU, and do not halt during waitstate hold.

Although detailed instruction timing calculations can be performed using the tables from the previous sections, it is usually more convenient to use the CPI figures to derive an approximate performance figure for a given ERC32 application. As an example, an application with 10% single precision floating-point instructions would have an average CPI of 0.9 * 1.4 + 0.1 * 2.5 = 1.51. At 14 MHz, the performance would be 14.0 / 1.51 = 9.27 MOPS.