

# The What and Why of System z Millicode

St. Louis CMG Local Users Group  
July, 2016

Bob Rogers  
rrrogers@us.ibm.com

## What is Millicode?

- Answer: Millicode is a form of microcode
- So what is microcode?
  - To paraphrase Wikipedia: **Microcode** is a layer of hardware-level instructions involved in the implementation of higher level machine code instructions.
  - A microarchitecture is the actual architecture of a machine which executes internal microcode which in turn implements the external architecture of the machine as seen by software.
  - Microcode is used not only to implement the instruction set architecture but also other controls like interrupt processing.

The invention of microcode is attributed to M.V.Wilkes, *Proc. Cambridge Phil. Soc.*, pt. 2, vol. 49, April 1953, pp. 230-238.

The term *architecture* was first applied to a computer instruction set by Gene Amdahl.

## What's good about Microcode?

- Originally developed as a simpler method of developing the control logic for a computer.
- Machines with differing internal design and built on different hardware technologies can present a single architecture to software.
- Logic errors in the microcode can be fixed with a reload of the microcode without actual hardware changes.
- Microcode patches can even be used to work around errors in the hardware.

## An Example: System/360 Model 30

- Most models of the IBM System/360 line were microcoded machines.
- This allowed the System/360 architecture to be provided on machines across wide ranges of price and performance based on the underlying technology and engineering.
- The Model 30 was the low-end model.
  - Implemented the 32-bit S/360 architecture on a much simpler machine:
    - An 8-bit machine with 8-bit data paths
    - GPRs were actually in core memory
    - The microcode itself was in a Read-only Store (ROS)

## Forms of microcode: Horizontal

- Horizontal microcode
  - A type of code in which the instructions are composed of a sequence of bit fields that more or less directly control the data flow within the CPU.
  - Horizontal microcode instructions are very literal and do not need much decoding.
  - A single micro-instruction might perform more than one operation in a single cycle.
    - As a simple example, one instruction might
      - Gate the value in a register to the left half of an adder
      - Gate the value in another register to the right half of the adder
      - Cause a 2s-compliment addition in the adder
      - Shift the result by 4 bits
      - Gate the result to some other register
    - VLIW (very long instruction word) architectures are somewhat similar to a horizontal micro-architecture

## Forms of microcode: Vertical

- Vertical microcode
  - A type of code with instructions very similar to the types of instructions that programmers are familiar with.
  - Basically, a vertical micro-architecture is just the architecture of a simpler machine than the one presented to programmers.
  - On the simpler machine, the hardware and microcode implement the more complex architecture which is presented to programmers.
  - Each micro-instruction typically does only a single operation in a cycle.

## An Example: S/390 9672 G3

- The early CMOS S/390 processors used vertical microcode. It was very similar to familiar machine language instructions, but the architecture is much simpler than the target architecture (i.e. S/390).
- Most of the frequently used simple instructions were implemented directly in hardware.
- More complex functions were implemented in vertical microcode that ran on distinct microcode processors which acted somewhat like a coprocessor.
- The microcode programs were stored in a special memory. If there was more microcode than fit into the special memory then the microcode for some instructions had to be "paged in" from regular memory before execution.



## Forms of microcode: Millicode

- Millicode on the IBM mainframe is a vertical microcode with some differences
- IBM System z micro-architecture instruction set is a superset of a subset of the instruction set of the external architecture described in *Principles of Operation*.
  - Millicode runs on the same hardware processor as customer software.
  - Architected instructions that are implemented in hardware are also available to millicode programs.
  - The millicode architecture includes additional instructions and registers not available in the external architecture.
  - Instructions which are implemented in millicode, of course, are not available to millicode programs



## An Example: S/390 9672 G4

- Starting with 9672 G4, the processors used a vertical code which is very similar to the target architecture (S/390, z/Architecture).
- It is so similar that it is executed on the same processor as the target architecture. There is no need for a special microprocessor.
- The programs are stored in standard memory (but not accessible to programs) and accessed via the L1 instruction cache, just like normal programs. There is no special instruction store<sup>1</sup>.
- Because of these similarities to normal code, this type of code was named **Millicode**.

<sup>1</sup>Actually, the G4 had a read only millicode cache to hold some performance sensitive routines. The rest of the millicode was handled as described.

## Millicode Implementation

- Since millicode executes on the same processor as zArchitecture instructions, that processor must be augmented with additional state information and be capable of executing additional instruction types.
  - there is a "millicode mode" - millimode
  - millicode status: milli-GPRs, milli-ARs, milli-CRs, millicode instruction address register plus other millicode registers
  - there are instructions to move data between the architected register and the millicode registers.
  - other special instructions only available in millimode
  - cannot use instruction implemented in millicode
  - when millicode is entered, specific millicode registers are loaded before the millicode is given control.

## Uses of System z Millicode

- On modern systems, most of z/Architecture is implemented in hardware.
- Millicode augments the hardware to provide:
  - System configuration functions
  - System initialization functions
  - Virtualization support for logical partitioning
  - Complex instructions
  - I/O functions
  - Interrupts and other control functions
  - RAS, Recovery, Logouts
  - Instrumentation

## Reasons for Millicoded Instructions

- Very complex instructions can be implemented with reasonable engineering expense.
  - Conversely, the ability to implement in millicode allows the architecture the freedom to define very complex instructions.
- Compatibility across processor generations can be provided by an early implementation in millicode.
- Millicode has access to hardware facilities not available to normal code.

## More Reasons for Millicoded Instructions

- Millicoded instructions are non-interruptible<sup>1</sup>.
  - Can perform multiple storage updates without requiring explicit disablement
- Millicode runs with high authorization.
  - This allows well-defined operations to be performed without needing to invoke the operating system (e.g. via SVC).
- A millicoded instruction can provide common functions almost like subroutines.

<sup>1</sup>Actually, some millicoded instructions are interruptible, but under strict millicode control

## Very Complex Millicoded Instructions

- Using millicode allows engineers to provide complex functions without complex logic design.
  - Many of the most complex z/Architecture instruction include over 100 cycles of activity
  - Examples:
    - Program Call (PC), Branch and Stack (BAKR), Program Transfer (PT)
    - Load Address Space Parameters (LASP)
    - Perform Locked Operation (PLO)
    - Cipher Message (KM)
    - I/O Instructions (SSCH, TSCH, HSCH, TPI), etc.

## Compatibility through Millicode Implementation

- Instructions and processor facilities can be provided earlier by providing an initial implementation in millicode.
- This enables greater compatibility and provides a test environment for functions later implemented in hardware.
- Examples:
  - Compare Double and Swap (64) on z900
  - Long Displacement Facility on z900
  - Decimal Floating Point on z990



## Millicoded Instructions as Subroutines

- Some millicoded instructions are implemented to provide a common function.
  - MVCL and CLCL are examples of functions that can be done in normal code in a dozen lines of code or so (more if padding).
  - Others include Translate (TR), Translate and Test (TRT).
  - In many cases, implementing these functions in normal code actually takes less cycles than using the millicoded instructions.
  - However, in some cases, there is special hardware to make these common functions execute more efficiently.
  - Using these “machine-provided subroutines” can reduce code bloat and improve programmer productivity.

## Millicoded Instructions for non-interruptibility

- Millicode can efficiently implement functions that must execute without interruption.
  - Millicode, by its nature, is not interruptible.
  - Using a millicoded instruction is faster than disabling for interrupts in normal code, performing the function and then re-enabling.
  - Disabling for interrupts is not available to unauthorized application code. Unauthorized code must depend upon an operating system service to perform the function.
  - An example is Extract CPU Time (ECTR). It “atomically” extracts the CPU Timer value, subtracts it for one operand and adds a second operand to the result.
    - An interruption during these calculations leads to incorrect results because the operating system updates the fields which are the operands whenever an interrupt occurs.
    - Before this instruction was introduced, an operating system service was needed to do this calculation of “CPU time used”.
  - The Perform Locked Operation (PLO) is an obvious example of taking advantage of the non-interruptibility of millicode.

## Millicoded Instructions for Authority

- Millicode can perform operations that require levels of authority
  - Millicode is self-enforcing as far as authorization is concerned – it does whatever it wants to do.
  - Millicode can use its authority to perform well-defined operation for unauthorized programs.
  - Using a millicoded instruction is faster than using an SVC or PC routine that has the appropriate authority.
  - Examples are the stacking instructions and Extract CPU Time.

## Millicode for access to special hardware

- In some cases, millicode has access to hardware facilities not available to normal code.
- For example, the MVCL instruction can use a hardware “data mover” when moving a full 4K frame of data (i.e. 4K bytes on a 4K boundary).
- The “data mover” can move data out in the memory nest without bringing it into processor cache. The data can be moved as a number of blocks that can be moved in parallel.
- As another example, the Edit (ED) and Edit and Mark (EDMK) instructions are supported by special hardware not available in the external architecture.

## Performance of Millicoded Instructions

- Millicode runs in the same way and at the same speed as ordinary software
- Therefore, it's often possible to implement a function in normal code which runs faster than using a millicoded instruction to do that function.
- When there is a choice of using a millicoded instruction or writing the function in open code, trade-offs need to be considered.
  - Convenience and programmer productivity
  - Code footprint size
  - Whether non-interruptibility is required
  - Whether the millicode has access to special hardware (e.g. some cases of MVCL and CLCL, data compression, encryption)

## Why Millicode is a Great Idea

- Millicode uses the same processor hardware as ordinary software.
- This avoids additional cost to design and manufacture a separate processor to run millicode as is often the case with traditional vertical microcode.
  - Millicode performance benefits from all the optimizations of the main processing engine.
  - Millicode RAS benefits from all the RAS mechanisms of the main processing engine.
  - Very low latency in switching into and out of millicode execution.



## How Millicode differs from Program Code

- To get its work done,
  - Millicode needs additional hardware registers:
    - Millicode GPRs, ARs, millicode instruction address register
    - Special registers: Operand Access Control Registers (OACRs)
  - Special instructions:
    - To copy values between the architected registers and the millicode registers
    - Instructions to access special hardware
  - Millicode needs to be able to access data
    - in memory in the current partition using application addressability
    - in system memory that is not in any partition.



## Millicode GPRs and ARs

- Millicode has its own 16 GPRs and 16 ARs.
- Operands in program registers need to be transferred to millicode registers and results returned to program registers.
- Some operand registers are specified in the instruction text and some are implicit.
- There are special millicode instructions to extract or set program registers.
  - Extract Program GR, Set Program GR
  - Extract Program GR Indirect, Set Program GR Indirect
  - Extract Program AR, Set Program AR
  - Extract Program AR Indirect, Set Program AR Indirect
- To assist with transferring the data, four 4-bit register indirect tags are defined, with each pointing to the GR specified as an (implicit or explicit) operand by the instruction.

## Millicode Startup

- Even for instructions implemented in millicode, the hardware performs some setup before branching to the millicode routine.
- For example, here is the setup for Compare Until Substring Equal (CUSE):
  - Machine Check if Millicode Mode
  - Specification Exception if  $R_1$  or  $R_2$  are Odd GRs
  - $IAREGA7_{0:31}$  set to Instruction Text
  - $RI_0$  set to  $R_1$  GR number
  - $RI_1$  set to  $R_2$  GR number
  - $RI_2$  set to  $R_{1+1}$  GR number
  - $RI_3$  set to  $R_{2+1}$  GR number

## Millicode Memory Access

- When a GPR is used as a base register, the register number is significant in determining the location of the data.
  - Registers 1-7: the storage access is made using the same addressing mode that is currently indicated by the system program.
  - Registers 12-15: the corresponding address is treated as a hardware system area address.
  - Register of 8-11: special hardware designates the address mode used for the storage access.
  - Four OACRs correspond one-to-one to millicode base registers 8-11. These registers include:
    - storage access key
    - address-space control (primary, secondary, home, or access register)
    - addressing mode (24-bit, 31-bit, or 64-bit addressing)
    - addressing type (real, virtual, host real, absolute, hw system area)
    - special controls which can block program event recording (PER) storage alteration detection or protection exceptions, and can pretest for store-type access exceptions.

## Perform Translator Operation (PXLO)

- *Load Address Space Control Element* - Determines the ASCE used for a translation
- *Load Absolute Address* - Obtains an Absolute Address of a translation
- *Load Real Address* - Obtains the Real Address of a translation
- *Load Host Real Address* - Used while in emulation mode to obtain the Host Real Address, when translating a Host Virtual Address
- *Load Page Table Entry* - Obtains the Page Table Entry address for a translation
- *Load Host Page Table Entry* - Used while in emulation mode to obtain the address of the Host Page Table Entry, when translating a Host Virtual Address
- *Purge TLB* - Purges previous translations from the local TLB
- *Invalidate Page Table Entry* - Invalidates selected entries from the local TLB
- *Purge Data Cache* - Purges all entries from the Data Cache
- *Purge Instruction Cache* - Purges all entries from the Instruction Cache

Adapted from *Millicode in an IBM zSeries processor*, IBM Journal of Research and Development, Volume 48 Issue 3-4, May 2004

## A Footnote to History

- The floppy disk was invented specifically as a way of loading microcode.
- In about 1971 IBM started using floppy disks as a medium for loading microcode into their System/370 computers during  $\mu$ IPL prior to system IPL.
- These were 8 inch floppy disks.

## Bibliography

- *Millicode in an IBM zSeries processor*, IBM Journal of Research and Development, Volume 48 Issue 3-4, May 2004
- *Even More of What You Do When You're a CPU?* SHARE 105 (Boston) session 2835 by Bob Rogers
- *Coding Assembler for Performance*, SHARE 107 (Baltimore) session 8192 by David Bond