ACA HOMEWORK #4

1)

In this paper the author presents an architectural view on VLIW processors to speed up the computers. In the VLIW compiler presents many operations which are to be processed at once. As compared to the regular process the performance is improved because the VLIW issues many of the process in a single execution stream rather than issuing a single operation from a single stream in a given cycle. In VLIW the compiler rearranges the program in advance, and selecting which issue to addressed at what time, to enhance the parallelism and maintaining the correct behaviour of the process, while this is done by the hardware in other processors. Therefore, the performance achieved by the VLIW is higher with a very low silicon and the power costs. The architecture of a VLIW is made up of functional units, registers and paths to memory. The beneficial factor of using the VLIW architectures is that the designers have the freedom to make changes and scale to various sizes. The first VLIW computers which were developed were the first mini-supercomputers.

a)

The applications which would benefit most from the VLIW architectures are high performance embedded computing applications, where the VLIW application had been dominating the digital video and the cellular base stations and playing a significant role in the media- intensive cell phones and personal digital assistants. Wherever high performance computing has been implemented VLIW, has become the architectural style of choice. The other common examples where VLIW architectures are used in digital televisions, digital cameras, printing, copying, faxing and scanning.

b)

There had been attempts in 1990 to integrate the VLIW architecture for the general purposes. Intel had launched the itanium processor which had design and characteristics similar to the VLIW. Transmeta company tried to emulate the Intel x86 by the VLIW processor. Although this had a positive impact on the cost/performance characteristics, those were not able to withstand the intel strength in the market.

2)

In this paper the author presents the concept of Simultaneous multithreading which would create a potential platform for the next generation computers, i.e to increase the processors computational capabilities which means to increase the parallelism of the process by using simultaneous multithreading.

a)

before adding the SMT the computer made the assumptions that there could have been a billion of transistors on the chip, so that the computer architects could add more memory, but this had a backdrop in the overall performance of the processor due to the memory wall constraint. The assumptions made about the computer architecture were:

* there were an eight-instruction fetch/decode width.
* There are a total of 32 integer and floating point dispatch queue.
* there were 6 integer units and the 4 integer units among them were to store in memory and vice versa.
* the number of floating point units were 4
* McFarling- a style branch prediction hardware.
* the hardware had a context of 8 threads.
* there were a 100 each of additional integer and floating point integers.
* 128-bit bus has a 80 cycle memory latency
* 16MB, direct mapped, unified L2 cache-the single.

b)

The Wall stated that ILP is limited to a maximum of 4 instructions that can be parallelized out of 10. Due to this limitation in the code, Wall introduces SMT which provides a better solution to the above limitation, by designing a processor which could exploit all levels of the parallelism. In his paper presented the results by raising the throughput to 6.2 for multiprogramming workload and 6.1 for the parallel programming.

c)

The SMT architecture was derived from the super scalar architecture. The SMT can execute up to 8 threads or more of the hardware contexts. The factors which were influential to replicate the SMT model were:

• State for the hardware contexts such as registers and program counters.

• Per thread identifiers to the branch target buffer and translation look aside buffer.

The changes which were made to the SMT were the instruction fetch and the processor pipeline

Instruction fetch:

The instruction fetches instructions from a single thread into the execution unit . A 2.8 scheme is implemented where every unit has a 8 program counter, 1 for each thread context. A feedback mechanism is also implemented for the thread selection to provide the priority to the threads with fewest instructions in the decode, renaming and queuing pipeline stages.

Processor Pipeline:

The pipeline capacity of SMT is large and also has an extended processor cycle time. The pipeline has 2 stages which are extended to allow 2 cycle register reads and writes

d)

The performance advantages claimed by the SMT were

* SMT processors had achieved a higher performance than other design architectures such as superscalar, traditional multithreaded and on chip multiprocessor
* SMT had a rise in throughput to 6.2 for multi-programming workload and 6.1 for the parallel workload.
* The SMT has sped up the parallel applications to 8 threads averaged to 1.9 over the same processor with one thread.
* For a single thread instruction throughput was 2.7 instruction for one cycle and 3.3 for the parallel instruction per cycle, whereas SMT had executed multiprogramming workload 2.3 times faster and parallel workload 1.9 times faster.
* SMT had a higher instruction throughput and greater program speedups

----------------------------------------------------------------------------------------------------------------

3)

a) The cluster architecture is the most dominant architecture which has a share of 86.3% system and 56.8% performance share. There are 456 counts of this architecture implementation

The most dominant operating system is LINUX which has 58.8% to operating system share and 39.4% operating system performance share. There are 337 counts of this operating system architecture.

b)

|  |  |  |
| --- | --- | --- |
| **System** | **Sunway TaihuLight** | **Tianhe-2 (MilkyWay-2)** |
| **Architecture** | Sunway MPP | TH-IVB-FEP Cluster |
| **Processor** | Sunway SW26010 260C 1.45GHz | Intel Xeon E5-2692 12C 2.200GHz |
| **Operating System** | Sunway RaiseOS 2.0.5 | Kylin Linux |
| **Interconnect** | Sunway | TH Express-2 |
| **Cores** | 10,649,610 | 3,120,000 |
| **Rmax (TFlop/s)** | 93,015.6 | 33,864.7 |
| **Rpeak (TFlop/s)** | 125,435.9 | 54,902.4 |
| **Power (kW)** | 15,371 | 17,808 |
| **Memory** | 1,310,720 GB | 1,024,000 GB |

----------------------------------------------------------------------------------------------------------------

Exercises

1) ADDI R1,R1,R1

ADDI R1,R1,R1

ADDI R1,R1,R1

R1 starts with 5

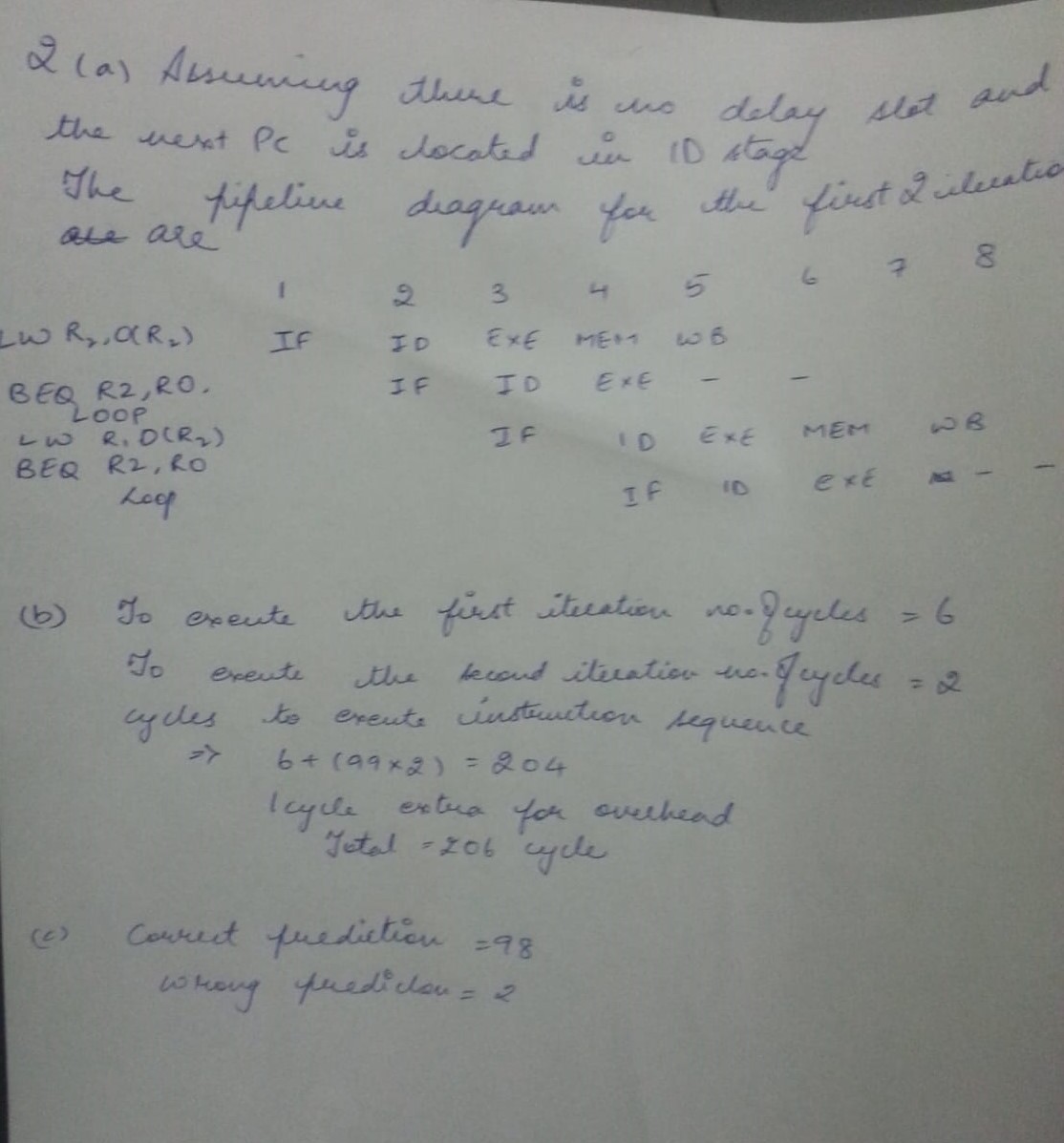
ADDI R1,R1,R1 =>R1=R1+R1=5+5=10

ADD R1,R1,R1 =>R1=R1+R1=10+10=20

ADD R1,R1,R1 =>R1=R1+R1=20+20=40

VALUE OF R1 after execution is 40

----------------------------------------------------------------------------------------------------------------

2) 

----------------------------------------------------------------------------------------------------------------

s3)

The prediction is mainly involved of using branch history tables and the prediction buffers. The branch prediction is used to predict if the branch is taken at the compile time or dynamically. Therefore the embedded processors which are small and less powerful would not be appropriate for the purpose of prediction and speculation.

Markets using the speculative prediction have some constraints on the resources which are to be used and this causes to have a known pattern of the power, cost and the processors, and thus these are the ones in which the branch prediction and speculation is less values.

----------------------------------------------------------------------------------------------------------------

4) There are 6 dependencies in this loop

True dependency from S1 to S3 on a[]

True dependency from S1 to S2 on a[]

Anti-dependency from S1 to S2 on b[]

Loop carried true dependency from S3 to S2 on a[]

Loop carried true dependency from S3 to S3 on a[]

Loop carried output dependency from S3 to S1 on a[]

To make the code parallel , we need to remove the output dependencies using renaming.

Loop carried dependencies are removed by changing the structure of the code.

Unrolling the code we get

a[0]=b[0]+c[0];

b[0]=a[0]+d[0];

a[1]=a[0]+e[0];

a[1]=b[1]+c[1];

b[1]=a[1]+d[1];

a[2]=a[1]+e[1];

a[2]=b[2]+c[2];

b[2]=a[2]+d[2];

a[3]=a[2]+e[2];

Now we can write the statement S3 outside the loop and rewrite the loop as follows

for(i=1;i<=99;i=i+1)

{

a[i]=b[i]+c[i]; /\*S1\*/

b[i]=a[i]+d[i]; /\*S2\*/

}

a[100]=a[99]+e[99];

----------------------------------------------------------------------------------------------------------------

LAB

5)

.file "example.c"

.text

.Ltext0:

.globl a

.data

.align 4

.type a, @object

.size a, 4

a:

.long 1

.globl b

.align 4

.type b, @object

.size b, 4

b:

.long 2

.globl c

.align 4

.type c, @object

.size c, 4

c:

.long 3

.text

.globl proc1

.type proc1, @function

proc1:

.LFB0:

.file 1 "/tmp/compiler-explorer-compiler11831-62-w4qe2u.s6paf/example.c"

.loc 1 3 0

.cfi\_startproc

pushq %rbp

.LCFI0:

.cfi\_def\_cfa\_offset 16

.cfi\_offset 6, -16

movq %rsp, %rbp

.LCFI1:

.cfi\_def\_cfa\_register 6

movl %edi, -4(%rbp)

movl %esi, -8(%rbp)

movl %edx, -12(%rbp)

.loc 1 4 0

cmpl $0, -4(%rbp)

jle .L2

.loc 1 6 0

movl -12(%rbp), %eax

movl -8(%rbp), %edx

movl %edx, %ecx

subl %eax, %ecx

movl %ecx, %eax

jmp .L3

.L2:

.loc 1 8 0

movl -12(%rbp), %eax

movl -8(%rbp), %edx

addl %edx, %eax

.L3:

.loc 1 9 0

popq %rbp

.LCFI2:

.cfi\_def\_cfa 7, 8

ret

.cfi\_endproc

.LFE0:

.size proc1, .-proc1

.globl main

.type main, @function

main:

.LFB1:

.loc 1 11 0

.cfi\_startproc

pushq %rbp

.LCFI3:

.cfi\_def\_cfa\_offset 16

.cfi\_offset 6, -16

movq %rsp, %rbp

.LCFI4:

.cfi\_def\_cfa\_register 6

subq $16, %rsp

.loc 1 13 0

movl c(%rip), %edx

movl b(%rip), %ecx

movl a(%rip), %eax

movl %ecx, %esi

movl %eax, %edi

call proc1

movl %eax, -8(%rbp)

.loc 1 14 0

movl -8(%rbp), %eax

cmpl $1, %eax

je .L7

cmpl $2, %eax

je .L8

testl %eax, %eax

jne .L11

.loc 1 16 0

movl $3, -4(%rbp)

.loc 1 17 0

jmp .L9

.L7:

.loc 1 18 0

movl $4, -4(%rbp)

.loc 1 19 0

jmp .L9

.L8:

.loc 1 20 0

movl $7, -4(%rbp)

.loc 1 21 0

jmp .L9

.L11:

.loc 1 22 0

movl $9, -4(%rbp)

.loc 1 23 0

nop

.L9:

.loc 1 25 0

movl -4(%rbp), %eax

movl -8(%rbp), %edx

addl %edx, %eax

movl %eax, -12(%rbp)

movl $0, %eax

.loc 1 26 0

leave

.LCFI5:

.cfi\_def\_cfa 7, 8

ret

.cfi\_endproc

.LFE1:

.size main, .-main

.Letext0:

.section .debug\_info,"",@progbits

.Ldebug\_info0:

.long 0xfe

.value 0x2

.long .Ldebug\_abbrev0

.byte 0x8

.uleb128 0x1

.long .LASF0

.byte 0x1

.long .LASF1

.quad .Ltext0

.quad .Letext0

.long .Ldebug\_line0

.uleb128 0x2

.byte 0x1

.long .LASF2

.byte 0x1

.byte 0x2

.byte 0x1

.long 0x74

.quad .LFB0

.quad .LFE0

.long .LLST0

.byte 0x1

.long 0x74

.uleb128 0x3

.string "a"

.byte 0x1

.byte 0x2

.long 0x74

.byte 0x2

.byte 0x91

.sleb128 -20

.uleb128 0x3

.string "b"

.byte 0x1

.byte 0x2

.long 0x74

.byte 0x2

.byte 0x91

.sleb128 -24

.uleb128 0x3

.string "c"

.byte 0x1

.byte 0x2

.long 0x74

.byte 0x2

.byte 0x91

.sleb128 -28

.byte 0

.uleb128 0x4

.byte 0x4

.byte 0x5

.string "int"

.uleb128 0x5

.byte 0x1

.long .LASF3

.byte 0x1

.byte 0xa

.long 0x74

.quad .LFB1

.quad .LFE1

.long .LLST1

.byte 0x1

.long 0xc5

.uleb128 0x6

.string "w"

.byte 0x1

.byte 0xc

.long 0x74

.byte 0x2

.byte 0x91

.sleb128 -24

.uleb128 0x6

.string "x"

.byte 0x1

.byte 0xc

.long 0x74

.byte 0x2

.byte 0x91

.sleb128 -20

.uleb128 0x6

.string "y"

.byte 0x1

.byte 0xc

.long 0x74

.byte 0x2

.byte 0x91

.sleb128 -28

.byte 0

.uleb128 0x7

.string "a"

.byte 0x1

.byte 0x1

.long 0x74

.byte 0x1

.byte 0x9

.byte 0x3

.quad a

.uleb128 0x7

.string "b"

.byte 0x1

.byte 0x1

.long 0x74

.byte 0x1

.byte 0x9

.byte 0x3

.quad b

.uleb128 0x7

.string "c"

.byte 0x1

.byte 0x1

.long 0x74

.byte 0x1

.byte 0x9

.byte 0x3

.quad c

.byte 0

.section .debug\_abbrev,"",@progbits

.Ldebug\_abbrev0:

.uleb128 0x1

.uleb128 0x11

.byte 0x1

.uleb128 0x25

.uleb128 0xe

.uleb128 0x13

.uleb128 0xb

.uleb128 0x3

.uleb128 0xe

.uleb128 0x11

.uleb128 0x1

.uleb128 0x12

.uleb128 0x1

.uleb128 0x10

.uleb128 0x6

.byte 0

.byte 0

.uleb128 0x2

.uleb128 0x2e

.byte 0x1

.uleb128 0x3f

.uleb128 0xc

.uleb128 0x3

.uleb128 0xe

.uleb128 0x3a

.uleb128 0xb

.uleb128 0x3b

.uleb128 0xb

.uleb128 0x27

.uleb128 0xc

.uleb128 0x49

.uleb128 0x13

.uleb128 0x11

.uleb128 0x1

.uleb128 0x12

.uleb128 0x1

.uleb128 0x40

.uleb128 0x6

.uleb128 0x2117

.uleb128 0xc

.uleb128 0x1

.uleb128 0x13

.byte 0

.byte 0

.uleb128 0x3

.uleb128 0x5

.byte 0

.uleb128 0x3

.uleb128 0x8

.uleb128 0x3a

.uleb128 0xb

.uleb128 0x3b

.uleb128 0xb

.uleb128 0x49

.uleb128 0x13

.uleb128 0x2

.uleb128 0xa

.byte 0

.byte 0

.uleb128 0x4

.uleb128 0x24

.byte 0

.uleb128 0xb

.uleb128 0xb

.uleb128 0x3e

.uleb128 0xb

.uleb128 0x3

.uleb128 0x8

.byte 0

.byte 0

.uleb128 0x5

.uleb128 0x2e

.byte 0x1

.uleb128 0x3f

.uleb128 0xc

.uleb128 0x3

.uleb128 0xe

.uleb128 0x3a

.uleb128 0xb

.uleb128 0x3b

.uleb128 0xb

.uleb128 0x49

.uleb128 0x13

.uleb128 0x11

.uleb128 0x1

.uleb128 0x12

.uleb128 0x1

.uleb128 0x40

.uleb128 0x6

.uleb128 0x2116

.uleb128 0xc

.uleb128 0x1

.uleb128 0x13

.byte 0

.byte 0

.uleb128 0x6

.uleb128 0x34

.byte 0

.uleb128 0x3

.uleb128 0x8

.uleb128 0x3a

.uleb128 0xb

.uleb128 0x3b

.uleb128 0xb

.uleb128 0x49

.uleb128 0x13

.uleb128 0x2

.uleb128 0xa

.byte 0

.byte 0

.uleb128 0x7

.uleb128 0x34

.byte 0

.uleb128 0x3

.uleb128 0x8

.uleb128 0x3a

.uleb128 0xb

.uleb128 0x3b

.uleb128 0xb

.uleb128 0x49

.uleb128 0x13

.uleb128 0x3f

.uleb128 0xc

.uleb128 0x2

.uleb128 0xa

.byte 0

.byte 0

.byte 0

.section .debug\_loc,"",@progbits

.LLST0:

.quad .LFB0-.Ltext0

.quad .LCFI0-.Ltext0

.value 0x2

.byte 0x77

.sleb128 8

.quad .LCFI0-.Ltext0

.quad .LCFI1-.Ltext0

.value 0x2

.byte 0x77

.sleb128 16

.quad .LCFI1-.Ltext0

.quad .LCFI2-.Ltext0

.value 0x2

.byte 0x76

.sleb128 16

.quad .LCFI2-.Ltext0

.quad .LFE0-.Ltext0

.value 0x2

.byte 0x77

.sleb128 8

.quad 0

.quad 0

.LLST1:

.quad .LFB1-.Ltext0

.quad .LCFI3-.Ltext0

.value 0x2

.byte 0x77

.sleb128 8

.quad .LCFI3-.Ltext0

.quad .LCFI4-.Ltext0

.value 0x2

.byte 0x77

.sleb128 16

.quad .LCFI4-.Ltext0

.quad .LCFI5-.Ltext0

.value 0x2

.byte 0x76

.sleb128 16

.quad .LCFI5-.Ltext0

.quad .LFE1-.Ltext0

.value 0x2

.byte 0x77

.sleb128 8

.quad 0

.quad 0

.section .debug\_aranges,"",@progbits

.long 0x2c

.value 0x2

.long .Ldebug\_info0

.byte 0x8

.byte 0

.value 0

.value 0

.quad .Ltext0

.quad .Letext0-.Ltext0

.quad 0

.quad 0

.section .debug\_line,"",@progbits

.Ldebug\_line0:

.section .debug\_str,"MS",@progbits,1

.LASF2:

.string "proc1"

.LASF0:

.string "GNU C 4.7.2"

.LASF3:

.string "main"

.LASF1:

.string "/tmp/compiler-explorer-compiler11831-62-w4qe2u.s6paf/example.c"

.ident "GCC: (GCC-Explorer-Build) 4.7.2"

.section .note.GNU-stack,"",@progbits

Code Optimization

To optimize the code we need to use the constant values for b+c and b-c because both of them are constant values. b-c is always -1 and we would not any if else statement.

We also would not need the switch statement because the x will always call the default value of 9

The updated code is

int a =1, b=2, c=3;

int main(){

int w,x,y;

w = b - c;

switch(w) {

case 0: x=3; break;

case 1: x=4; break;

case 2: x=7; break;

default: x=9; break;

}

y = w+x;

return 0;

}

Assembly code after optimization

.file "example.cpp"

.text

.Ltext0:

.globl a

.data

.align 4

.type a, @object

.size a, 4

a:

.long 1

.globl b

.align 4

.type b, @object

.size b, 4

b:

.long 2

.globl c

.align 4

.type c, @object

.size c, 4

c:

.long 3

.text

.globl main

.type main, @function

main:

.LFB0:

.file 1 "/tmp/compiler-explorer-compiler11832-63-10cresd.j3zr/example.cpp"

.loc 1 2 0

.cfi\_startproc

pushq %rbp

.LCFI0:

.cfi\_def\_cfa\_offset 16

.cfi\_offset 6, -16

movq %rsp, %rbp

.LCFI1:

.cfi\_def\_cfa\_register 6

.LBB2:

.loc 1 4 0

movl b(%rip), %edx

movl c(%rip), %eax

movl %edx, %ecx

subl %eax, %ecx

movl %ecx, %eax

movl %eax, -8(%rbp)

.loc 1 5 0

movl -8(%rbp), %eax

cmpl $1, %eax

je .L4

cmpl $2, %eax

je .L5

testl %eax, %eax

jne .L8

.L3:

.loc 1 6 0

movl $3, -4(%rbp)

jmp .L6

.L4:

.loc 1 7 0

movl $4, -4(%rbp)

jmp .L6

.L5:

.loc 1 8 0

movl $7, -4(%rbp)

jmp .L6

.L8:

.loc 1 9 0

movl $9, -4(%rbp)

nop

.L6:

.loc 1 11 0

movl -4(%rbp), %eax

movl -8(%rbp), %edx

addl %edx, %eax

movl %eax, -12(%rbp)

.loc 1 12 0

movl $0, %eax

.LBE2:

.loc 1 13 0

popq %rbp

.LCFI2:

.cfi\_def\_cfa 7, 8

ret

.cfi\_endproc

.LFE0:

.size main, .-main

.Letext0:

.section .debug\_info,"",@progbits

.Ldebug\_info0:

.long 0xc5

.value 0x2

.long .Ldebug\_abbrev0

.byte 0x8

.uleb128 0x1

.long .LASF0

.byte 0x4

.long .LASF1

.quad .Ltext0

.quad .Letext0

.long .Ldebug\_line0

.uleb128 0x2

.byte 0x1

.long .LASF2

.byte 0x1

.byte 0x2

.long 0x85

.quad .LFB0

.quad .LFE0

.long .LLST0

.byte 0x1

.long 0x85

.uleb128 0x3

.quad .LBB2

.quad .LBE2

.uleb128 0x4

.string "w"

.byte 0x1

.byte 0x3

.long 0x85

.byte 0x2

.byte 0x91

.sleb128 -24

.uleb128 0x4

.string "x"

.byte 0x1

.byte 0x3

.long 0x85

.byte 0x2

.byte 0x91

.sleb128 -20

.uleb128 0x4

.string "y"

.byte 0x1

.byte 0x3

.long 0x85

.byte 0x2

.byte 0x91

.sleb128 -28

.byte 0

.byte 0

.uleb128 0x5

.byte 0x4

.byte 0x5

.string "int"

.uleb128 0x6

.string "a"

.byte 0x1

.byte 0x1

.long 0x85

.byte 0x1

.byte 0x9

.byte 0x3

.quad a

.uleb128 0x6

.string "b"

.byte 0x1

.byte 0x1

.long 0x85

.byte 0x1

.byte 0x9

.byte 0x3

.quad b

.uleb128 0x6

.string "c"

.byte 0x1

.byte 0x1

.long 0x85

.byte 0x1

.byte 0x9

.byte 0x3

.quad c

.byte 0

.section .debug\_abbrev,"",@progbits

.Ldebug\_abbrev0:

.uleb128 0x1

.uleb128 0x11

.byte 0x1

.uleb128 0x25

.uleb128 0xe

.uleb128 0x13

.uleb128 0xb

.uleb128 0x3

.uleb128 0xe

.uleb128 0x11

.uleb128 0x1

.uleb128 0x12

.uleb128 0x1

.uleb128 0x10

.uleb128 0x6

.byte 0

.byte 0

.uleb128 0x2

.uleb128 0x2e

.byte 0x1

.uleb128 0x3f

.uleb128 0xc

.uleb128 0x3

.uleb128 0xe

.uleb128 0x3a

.uleb128 0xb

.uleb128 0x3b

.uleb128 0xb

.uleb128 0x49

.uleb128 0x13

.uleb128 0x11

.uleb128 0x1

.uleb128 0x12

.uleb128 0x1

.uleb128 0x40

.uleb128 0x6

.uleb128 0x2117

.uleb128 0xc

.uleb128 0x1

.uleb128 0x13

.byte 0

.byte 0

.uleb128 0x3

.uleb128 0xb

.byte 0x1

.uleb128 0x11

.uleb128 0x1

.uleb128 0x12

.uleb128 0x1

.byte 0

.byte 0

.uleb128 0x4

.uleb128 0x34

.byte 0

.uleb128 0x3

.uleb128 0x8

.uleb128 0x3a

.uleb128 0xb

.uleb128 0x3b

.uleb128 0xb

.uleb128 0x49

.uleb128 0x13

.uleb128 0x2

.uleb128 0xa

.byte 0

.byte 0

.uleb128 0x5

.uleb128 0x24

.byte 0

.uleb128 0xb

.uleb128 0xb

.uleb128 0x3e

.uleb128 0xb

.uleb128 0x3

.uleb128 0x8

.byte 0

.byte 0

.uleb128 0x6

.uleb128 0x34

.byte 0

.uleb128 0x3

.uleb128 0x8

.uleb128 0x3a

.uleb128 0xb

.uleb128 0x3b

.uleb128 0xb

.uleb128 0x49

.uleb128 0x13

.uleb128 0x3f

.uleb128 0xc

.uleb128 0x2

.uleb128 0xa

.byte 0

.byte 0

.byte 0

.section .debug\_loc,"",@progbits

.Ldebug\_loc0:

.LLST0:

.quad .LFB0-.Ltext0

.quad .LCFI0-.Ltext0

.value 0x2

.byte 0x77

.sleb128 8

.quad .LCFI0-.Ltext0

.quad .LCFI1-.Ltext0

.value 0x2

.byte 0x77

.sleb128 16

.quad .LCFI1-.Ltext0

.quad .LCFI2-.Ltext0

.value 0x2

.byte 0x76

.sleb128 16

.quad .LCFI2-.Ltext0

.quad .LFE0-.Ltext0

.value 0x2

.byte 0x77

.sleb128 8

.quad 0

.quad 0

.section .debug\_aranges,"",@progbits

.long 0x2c

.value 0x2

.long .Ldebug\_info0

.byte 0x8

.byte 0

.value 0

.value 0

.quad .Ltext0

.quad .Letext0-.Ltext0

.quad 0

.quad 0

.section .debug\_line,"",@progbits

.Ldebug\_line0:

.section .debug\_str,"MS",@progbits,1

.LASF0:

.string "GNU C++ 4.7.2"

.LASF1:

.string "/tmp/compiler-explorer-compiler11832-63-10cresd.j3zr/example.cpp"

.LASF2:

.string "main"

.ident "GCC: (GCC-Explorer-Build) 4.7.2"

.section .note.GNU-stack,"",@progbits