Anybody who writes #pragma pack(1) may as well just wear a sign on their forehead that says “I hate RISC”

devblogs.microsoft.com/oldnewthing/20200103-00

January 3, 2020

Raymond Chen

When you use `#pragma pack(1)`, this changes the default structure packing to byte packing, removing all padding bytes normally inserted to preserve alignment.

Consider these two structures:

```
// no #pragma pack in effect.
struct S
{
    int32_t total;
    int32_t a, b;
};

#pragma pack(1)

struct P
{
    int32_t total;
    int32_t a, b;
};
```

Both structures have identical layouts because the members are already at their natural alignment, Therefore, you would expect these two structures to be equivalent.

But they’re not.

Changing the default structure packing has another consequence: It changes the alignment of the structure itself. In this case, the `#pragma pack(1)` declares that the structure `P` can itself be placed at any byte boundary, instead of requiring it to be placed on a 4-byte boundary.

```
struct ExtraS
{
    char c;
    S s;
    char d;
};

struct ExtraP
{
    char c;
    P p;
    char d;
};
```

Even though the structures `S` and `P` have the same layout, the difference in alignment means that the structures `ExtraS` and `ExtraP` end up quite different.

The `ExtraS` structure starts with a `char`, then adds three bytes of padding, followed by the `S` structure, then another `char`, and three more bytes of padding to bring the entire structure back up to 4-byte alignment. This ensures that an array of `ExtraS` structures will properly align all of the embedded `S` objects.

By comparison, the `ExtraP` structure starts out the same way, with a single `char`, but this time, there is no padding before the `P` because the `P` is byte-aligned. Similarly, there is no trail padding at the end of the structure because the `P` is byte-aligned and therefore does not need to be kept at a particular alignment in the case of an array of `ExtraP` objects.

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	
ExtraS	`c`	padding			`s.total`				`s.a`				`s.b`				`d`	
ExtraP	`c`	`p.total`				`p.a`				`p.b`					`d`			

The effect is more noticeable if you have an array of these objects.

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	
ExtraS	`c`	padding			`s.total`				`s.a`				`s.b`				`d`	
ExtraP	`c`	`p.total`				`p.a`				`p.b`				`d`	`c`	`p.total`		

Observe that in the array of `ExtraS` objects, the `s.total`, `s.a`, and `s.b` are always four-byte aligned. But in the array of `ExtraP` objects, there is no consistent alignment for the members of `p`.

The possibility that any `P` structure could be misaligned has significant consequences for code generation, because all accesses to members must handle the case that the address is not properly aligned.

```
void UpdateS(S* s)
{
 s->total = s->a + s->b;
}

void UpdateP(P* p)
{
 p->total = p->a + p->b;
}
```

Despite the structures `S` and `P` having exactly the same layout, the code generation is different because of the alignment.

UpdateS	UpdateP
Intel Itanium	

```
adds  r31 = r32, 4
adds  r30 = r32  8
;;
ld4   r31 = [r31]
ld4   r30 = [r30]
;;
```

```
adds  r31 = r32, 4
adds  r30 = r32  8 ;;
ld1   r29 = [r31], 1
ld1   r28 = [r30], 1 ;;
ld1   r27 = [r31], 1
ld1   r26 = [r30], 1 ;;
dep   r29 = r27, r29, 8,
8
dep   r28 = r26, r28, 8,
8
ld1   r25 = [r31], 1
ld1   r24 = [r30], 1 ;;
dep   r29 = r25, r29, 16,
8
dep   r28 = r24, r28, 16,
8
ld1   r27 = [r31]
ld1   r26 = [r30] ;;
```

```
add   r31 = r30,
r31 ;;
st4   [r32] = r31
```

```
dep   r29 = r27, r29, 24,
8
dep   r28 = r26, r28, 24,
8 ;;
add   r31 = r28, r29 ;;
st1   [r32] = r31
adds  r30 = r32, 1
adds  r29 = r32, 2
extr  r28 = r31, 8, 8
extr  r27 = r31, 16, 8 ;;
st1   [r30] = r28
st1   [r29] = r27, 1
extr  r26 = r31, 24, 8 ;;
st1   [r29] = r26
```

```
br.ret.sptk.many
rp
```

```
br.ret.sptk.many.rp
```

Alpha AXP

ldl t1, 4(a0)
 	
ldl t2, 8(a0)
 	
addl t1, t1, t2
stl t1, (a0)
 	
ret zero, (ra), 1
	ldq_u t1, 4(a0)
	ldq_u t3, 7(a0)
	extll t1, a0, t1
	extlh t3, a0, t3
	bis t1, t3, t1
	ldq_u t2, 8(a0)
	ldq_u t3, 11(a0)
	extll t2, a0, t2
	extlh t3, a0, t3
	bis t2, t3, t2
	addl t1, t1, t2
	ldq_u t2, 3(a0)
	ldq_u t5, (a0)
	inslh t1, a0, t4
	insll t1, a0, t3
	msklh t2, a0, t2
	mskll t5, a0, t5
	bis t2, t4, t2
	bis t5, t3, t5
	stq_u t2, 3(a0)
	stq_u t5, (a0)
	ret zero, (ra), 1

MIPS R4000	

lw t0, 4(a0)
 	
lw t1, 8(a0)
 	
addu t0, t0, t1
 	
jr ra
sw t0, (a0)
	lwl t0, 7(a0)
	lwr t0, 4(a0)
	lwl t1, 11(a0)
	lwr t1, 8(a0)
	addu t0, t0, t1
	swl t0, 3(a0)
	jr ra
	swr t0, (a0)

PowerPC 600	

`lwz r4, 4(r3)`
 	
`lwz r5, 8(r3)`
 	
`addu r4, r4, r5`
`stw r4, (r3)`
 	
`blr`
	`lbz r4, 4(r3)`
	`lbz r9, 5(r3)`
	`rlwimi r4, r9, 8, 16, 23`
	`lbz r9, 6(r3)`
	`rlwimi r4, r9, 16, 8, 15`
	`lbz r9, 7(r3)`
	`rlwimi r4, r9, 24, 0, 7`
	`lbz r5, 8(r3)`
	`lbz r9, 9(r3)`
	`rlwimi r5, r9, 8, 16, 23`
	`lbz r9, 10(r3)`
	`rlwimi r5, r9, 16, 8, 15`
	`lbz r9, 11(r3)`
	`rlwimi r5, r9, 24, 0, 7`
	`addu r4, r4, r5`
	`stb r4, (r3)`
	`rlwimi r9, r4, 24, 0, 31`
	`stb r9, 1(r3)`
	`rlwimi r9, r4, 16, 0, 31`
	`stb r9, 2(r3)`
	`rlwimi r9, r4, 8, 0, 31`
	`stb r9, 3(r3)`
	`blr`

SuperH-3	

`mov.l @(4, r4), r2`	`mov.b @(7, r4), r1`
	`shll8 r1`
	`mov.b @(6, r4), r2`
	`extu.b r2, r2`
	`or r2, r1`
	`shll8 r1`
	`mov.b @(5, r4), r2`
	`extu.b r2, r2`
	`or r2, r1`
	`shll8 r1`
	`mov.b @r4, r2`
	`extu.b r2, r2`
	`or r1, r2`

`mov.l @(8, r4), r3`	`mov.b @(7, r4), r1`
	`shll8 r1`
	`mov.b @(6, r4), r3`
	`extu.b r3, r3`
	`or r3, r1`
	`shll8 r1`
	`mov.b @(5, r4), r3`
	`extu.b r3, r3`
	`or r3, r1`
	`shll8 r1`
	`mov.b @r4, r3`
	`extu.b r3, r3`
	`or r1, r3`

`add r3, r2`	`add r3, r2`
	`mov.b r2, @r4`
	`shlr8 r2`
	`mov.b r2, @(1, r4)`
	`shlr8 r2`
	`mov.b r2, @(2, r4)`
	`shlr8 r2`

`rts` `mov.l r2, @r4`	`rts` `mov.b r2, @(3, r4)`

Observe that for some RISC processors, the code size explosion is quite significant. This may in turn affect inlining decisions.

Moral of the story: Don't apply `#pragma pack(1)` to structures unless absolutely necessary. It bloats your code and inhibits optimizations.

Bonus chatter: Once you make this mistake, you can't go back. You allowed the structure to be byte-aligned, and if you remove the spurious `#pragma pack(1)`, you are making the structure more strictly aligned, which will be a breaking change for any clients which used the byte-packed version.

Raymond Chen

Follow