```
**********************************************************
    4717 Wed Oct 14 16:45:06 2015
new/usr/src/lib/libdisasm/common/dis_i386.c
6068 libdisasm: previnstr arch op should have a sane default
Reviewed by: Robert Mustacchi <rm@joyent.com>
**********************************************************
______unchanged_portion_omitted_

175 /*
176  * Return the previous instruction.  On x86, we have no choice except to
177  * disassemble everything from the start of the symbol, and stop when we have
178  * reached our instruction address.  If we're not in the middle of a known
179  * symbol, then we return the same address to indicate failure.
180  */
181 static uint64_t
182 dis_i386_previnstr(dis_handle_t *dhp, uint64_t pc, int n)
183 {
184         uint64_t *hist, addr, start;
185         int cur, nseen;
186         uint64_t res = pc;

188         if (n <= 0)
189                 return (pc);

191         if (dhp->dh_lookup(dhp->dh_data, pc, NULL, 0, &start, NULL) != 0 ||
192             start == pc)
193                 return (res);

195         hist = dis_zalloc(sizeof (uint64_t) * n);

197         for (cur = 0, nseen = 0, addr = start; addr < pc; addr = dhp->dh_addr) {
198                 hist[cur] = addr;
199                 cur = (cur + 1) % n;
200                 nseen++;

202                 /* if we cannot make forward progress, give up */
203                 if (dis_disassemble(dhp, addr, NULL, 0) != 0)
204                         goto done;
205         }

207         if (addr != pc) {
208                 /*
209                  * We scanned past %pc, but didn't find an instruction that
210                  * started at %pc.  This means that either the caller specified
211                  * an invalid address, or we ran into something other than code
212                  * during our scan.  Virtually any combination of bytes can be
213                  * construed as a valid Intel instruction, so any non-code bytes
214                  * we encounter will have thrown off the scan.
215                  */
216                 goto done;
217         }

219         res = hist[(cur + n - MIN(n, nseen)) % n];

221 done:
222         dis_free(hist, sizeof (uint64_t) * n);
223         return (res);
224 }

175 static int
176 dis_i386_supports_flags(int flags)
177 {
178         int archflags = flags & DIS_ARCH_MASK;

180         if (archflags == DIS_X86_SIZE16 || archflags == DIS_X86_SIZE32 ||
181             archflags == DIS_X86_SIZE64)
```



```
182                 return (1);

184         return (0);
185 }
______unchanged_portion_omitted_

196 dis_arch_t dis_arch_i386 = {
197         .da_supports_flags      = dis_i386_supports_flags,
198         .da_handle_attach       = dis_i386_handle_attach,
199         .da_handle_detach       = dis_i386_handle_detach,
200         .da_disassemble         = dis_i386_disassemble,
252         .da_previnstr           = dis_i386_previnstr,
201         .da_min_instrlen        = dis_i386_min_instrlen,
202         .da_max_instrlen        = dis_i386_max_instrlen,
203         .da_instrlen            = dis_i386_instrlen,
204 };
______unchanged_portion_omitted_
```

****
8960 Wed Oct 14 16:45:06 2015
new/usr/src/lib/libdisasm/common/dis_sparc.c
6068 libdisasm: previnstr arch op should have a sane default
Reviewed by: Robert Mustacchi <rm@joyent.com>
****
_____***unchanged_portion_omitted_***

```
200 /*
201  * The dis_i386.c comment for this says it returns the previous instruction,
202  * however, I'm fairly sure it's actually returning the _address_ of the
203  * nth previous instruction.
204  */
200 /* ARGSUSED */
201 static uint64_t
202 dis_sparc_previnstr(dis_handle_t *dhp, uint64_t pc, int n)
203 {
204         if (n <= 0)
205                 return (pc);

207         if (pc < n)
208                 return (pc);

210         return (pc - n*4);
211 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    6523 Wed Oct 14 16:45:06 2015
new/usr/src/lib/libdisasm/common/libdisasm.c
6068 libdisasm: previnstr arch op should have a sane default
Reviewed by: Robert Mustacchi <rm@joyent.com>
**********************************************************
______unchanged_portion_omitted_
```

```
199 /*
200  * On some instruction sets (e.g., x86), we have no choice except to
201  * disassemble everything from the start of the symbol, and stop when we
202  * have reached our instruction address.  If we're not in the middle of a
203  * known symbol, then we return the same address to indicate failure.
204  */
205 static uint64_t
206 dis_generic_previnstr(dis_handle_t *dhp, uint64_t pc, int n)
207 {
208         uint64_t *hist, addr, start;
209         int cur, nseen;
210         uint64_t res = pc;
211
212         if (n <= 0)
213                 return (pc);
214
215         if (dhp->dh_lookup(dhp->dh_data, pc, NULL, 0, &start, NULL) != 0 ||
216             start == pc)
217                 return (res);
218
219         hist = dis_zalloc(sizeof (uint64_t) * n);
220
221         for (cur = 0, nseen = 0, addr = start; addr < pc; addr = dhp->dh_addr) {
222                 hist[cur] = addr;
223                 cur = (cur + 1) % n;
224                 nseen++;
225
226                 /* if we cannot make forward progress, give up */
227                 if (dis_disassemble(dhp, addr, NULL, 0) != 0)
228                         goto done;
229         }
230
231         if (addr != pc) {
232                 /*
233                  * We scanned past %pc, but didn't find an instruction that
234                  * started at %pc.  This means that either the caller specified
235                  * an invalid address, or we ran into something other than code
236                  * during our scan.  Virtually any combination of bytes can be
237                  * construed as a valid Intel instruction, so any non-code bytes
238                  * we encounter will have thrown off the scan.
239                  */
240                 goto done;
241         }
242
243         res = hist[(cur + n - MIN(n, nseen)) % n];
244
245 done:
246         dis_free(hist, sizeof (uint64_t) * n);
247         return (res);
248 }
249
250 /*
251  * Return the nth previous instruction's address.  Return the same address
252  * to indicate failure.
253  */
254 #endif /* ! codereview */
255 uint64_t
256 dis_previnstr(dis_handle_t *dhp, uint64_t pc, int n)
```



```
257 {
258         if (dhp->dh_arch->da_previnstr == NULL)
259                 return (dis_generic_previnstr(dhp, pc, n));
260
261 #endif /* ! codereview */
262         return (dhp->dh_arch->da_previnstr(dhp, pc, n));
263 }
264
265 int
266 dis_min_instrlen(dis_handle_t *dhp)
267 {
268         return (dhp->dh_arch->da_min_instrlen(dhp));
269 }
270
271 int
272 dis_max_instrlen(dis_handle_t *dhp)
273 {
274         return (dhp->dh_arch->da_max_instrlen(dhp));
275 }
276
277 int
278 dis_instrlen(dis_handle_t *dhp, uint64_t pc)
279 {
280         return (dhp->dh_arch->da_instrlen(dhp, pc));
281 }
282
283 int
284 dis_vsnprintf(char *restrict s, size_t n, const char *restrict format,
285     va_list args)
286 {
287 #ifdef DIS_STANDALONE
288         return (mdb_iob_vsnprintf(s, n, format, args));
289 #else
290         return (vsnprintf(s, n, format, args));
291 #endif
292 }
293
294 int
295 dis_snprintf(char *restrict s, size_t n, const char *restrict format, ...)
296 {
297         va_list args;
298
299         va_start(args, format);
300         n = dis_vsnprintf(s, n, format, args);
301         va_end(args);
302
303         return (n);
304 }
```