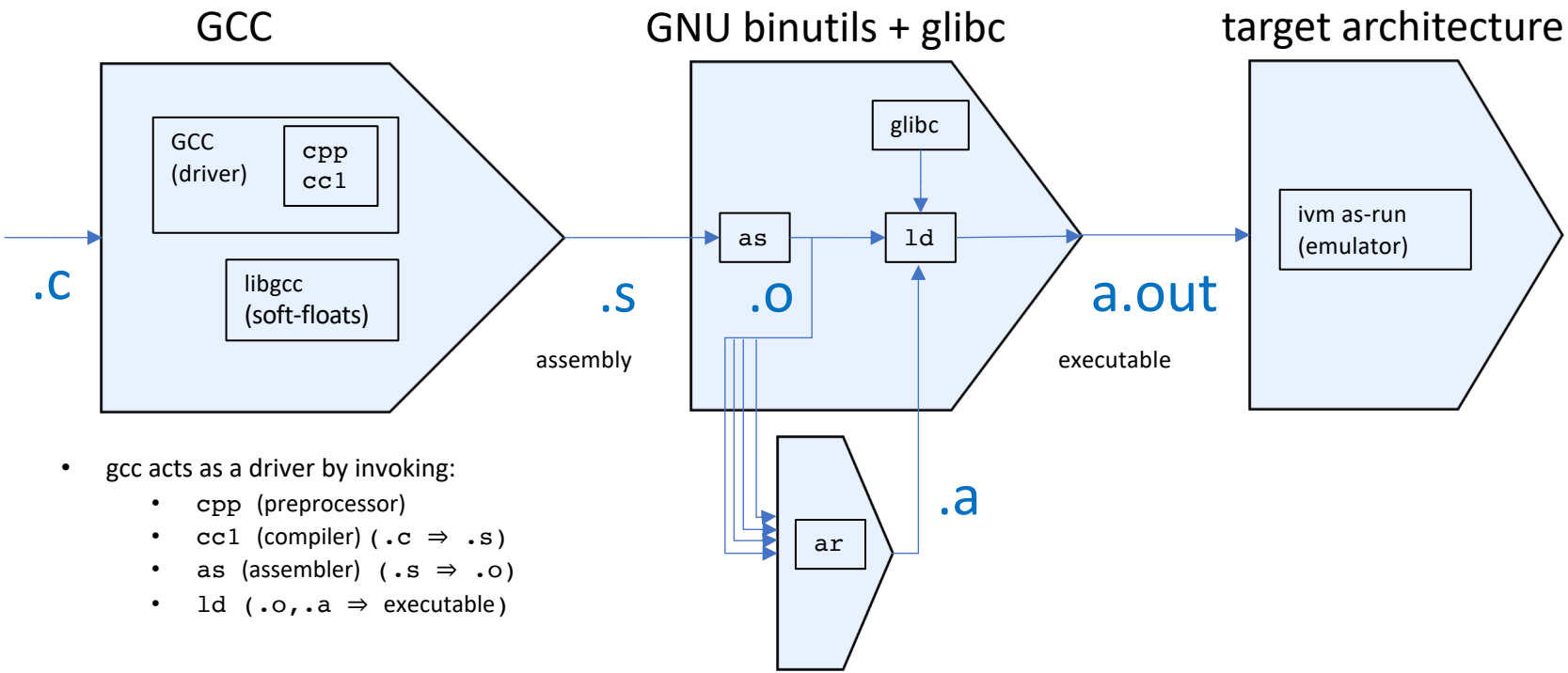
IVM64 Toolchain

with GNU GCC

February, 2020

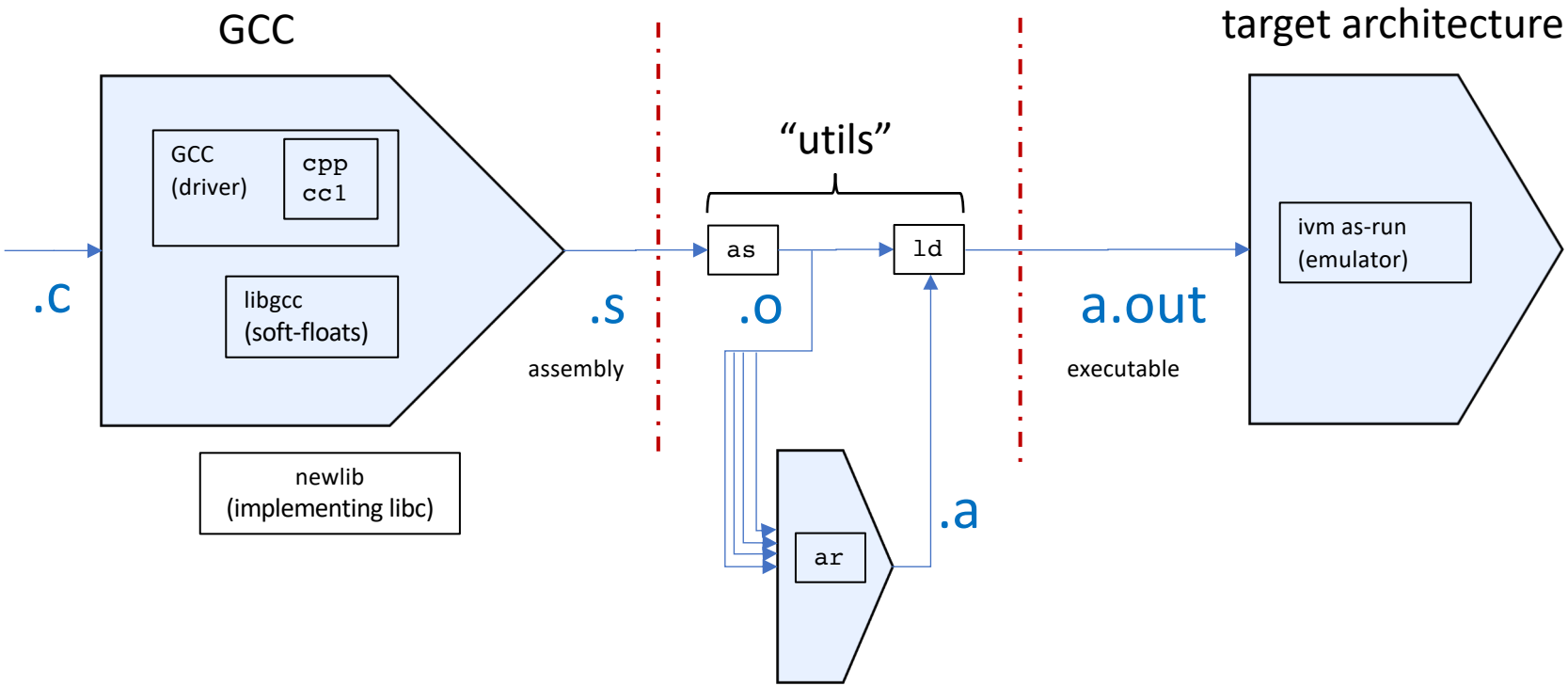
GNU Toolchain: GCC + binutils + glibc



- gcc acts as a driver by invoking:
  - `cpp` (preprocessor)
  - `cc1` (compiler) (`.c` ⇒ `.s`)
  - `as` (assembler) (`.s` ⇒ `.o`)
  - `ld` (`.o`,`.a` ⇒ executable)

- Libraries are created separately:
  - `ar` (`.o`,`.o`,`.o` ⇒`.a`)

IVM64 Toolchain: GCC + “other utils” + newlib



IVM64 Toolchain: GCC + “other utils”

- **`as` and `ld` must be provided with some minimal standard options**
- they can be scripts (wrappers) calling the assembler/linker programs
  - `as`:

    translates assembly to object (whatever an object is)
  - `ld`:

    links multiple libraries+objects (whatever objects are) into one single “executable” able to be processed by ”ivm” (or already processed)
- **libraries** pack multiple objects in one single file (but they are not linked, just packed)
  - **only static (.a)** libraries are considered (no shared objects, .so)
  - standard **`ar`** “archiving” command used (as no need to process objects, only to pack them)

IVM64 GCC + “other utils”

- **Main advantage** of this approach:
  - **compatibility with** other tools, minimizing the effort to adapt existing C projects (configure, makefile, …)
- `as` and `ld` constitute the interface:
  - hooks for external assemblers/linkers
- See example *20-make* in the Github repository
  - Makefile based project
  - Using UMA as/ld scripts for linking (provided into the gcc release)

File Extension Convention

Extension	Used for
.s	Assembly (as generated by: `ivm64-gcc -S`)
.S	Assembly (special assembly files like handwritten assembly, initialization, …)
.o	Object (as generated by: `ivm64-gcc -c`)
.a	Static library (as generated by: `ar cr file1.o file2.o …`)

as/ld Minimal Options

- GCC driver **expects** certain options when it invokes the assembler and linker

```
as –o object.o file.s
```

The last argument is the name of the assembly input file
The output object file needs to be set

```
ld [–o outfile] [–L libdir] [–lname] [–e entry] obj1.o obj2.o …
```

There can be several ‘`–L`’ and ‘`–l`’ options to specify different library directories and libraries, respectively. Note that ‘`–lname`’ denotes library ’`libname.a`’
Current directory is always included by default (-L.) although not set.
If no '-o' option, output file will be a name by default (a.out).

Example: UMA Linking System

- Linking at assembly level
  - An object (.o) simply contains its corresponding assembly file (.s)
  - A library (.a) is a collection of objects created with ‘ar’
  - Linker: links all objects/libraries into one single assembly file by rewriting local labels with a unique suffix per object
  - The result is a single assembly file (the ”executable”) that can be processed with the “ivm” tool
  - In addition, the output is a true executable that can be executed directly (by means of a #! first line calling “ivm as-run”)

Example: UMA Linking System

- gcc driver should pass the linker `ld` the object corresponding to `crt0.c` in charge of calling `main()` (as one more object)
- UMA linker prepends this `crt0` to the beginning of the final output, making the `_start` label the default entry point (the label before the first assembly instruction)
- `ld` script could support `-e` option to define a different entry point (UMA `ld` does not support this currently)

```
/* crt0.c */
extern void exit(int code);
extern int main(int argc, char *argv[]);
void _start()
{
      int ex = main(0, 0);
      exit(ex);
}
```

Example: UMA Linking System

- UMA linking system works well with assembly files generated by `ivm64-gcc`

- Generate several assembly files from C sources:

```
ivm64-gcc -S file1.c file2.c ...
```

- Compile generating objects from C sources:

```
ivm64-gcc -c file1.c file2.c ...
```

- Link several C sources/asm/objects in one unique "executable" assembly:

```
ivm64-gcc main.c mod1.c mod2.c obj1.o obj2.o asm1.S ... -o a.out.s
```

- Build a library by compiling the objects and then archiving them:

```
ivm64-gcc -c fun1.c fun2.c ...
ar cr libmy.a fun1.o fun2.o ...
```

- Both -L/-l flags can be used; for example, linking with libmy.a in /path/to:

```
ivm64-gcc main.c -L/path/to/ -lmy -o a.out.s
```

Example: UMA Linking System

- `as`/`ld` implemented as bash scripts based on well-known unix tools such as `grep, awk, sed`, …
- They perform ok for gcc generated assembly files
- Some restrictions posed to manually written assembly:

- One only instruction per line

Supported:

```
push! 3
push! 4
```

Not supported:

```
push! 3 push! 4
```

- Label/Alias definitions in one only line

Supported:

```
main:

label:
   push
```

Not supported:

```
main
     :
label: push
```

- Data declarations must be in one line

Supported:

```
data1 [0 0 0 0]
```

Not supported:

```
data1 [0 0
       0 0]
```